PSZ 19:16 (Pind. 1/07)

UNIVERSITI TEKNOLOGI MALAYSIA

DECLARATION OF THESIS / UNDERGRADUATE PROJECT PAPER AND COPYRIGHT

Author's full name : NG DER LI

Date of birth : 30-12-1985

Title : Development 5 Fingers Robot Hand Using PIC

Academic Session: 2008/2009

I declare that this thesis is classified as :

☐	**CONFIDENTIAL**	(Contains confidential information under the Official Secret Act 1972)*
☐	**RESTRICTED**	(Contains restricted information as specified by the organisation where research was done)*
☑	**OPEN ACCESS**	I agree that my thesis to be published as online open access (full text)

I acknowledged that Universiti Teknologi Malaysia reserves the right as follows :

1. The thesis is the property of Universiti Teknologi Malaysia.
2. The Library of Universiti Teknologi Malaysia has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by :

SIGNATURE

851230-01-5503
(NEW IC NO. /PASSPORT NO.)

Date : 13-MAY-2009

SIGNATURE OF SUPERVISOR

DR. IZZELDIN IBRAHIM MOHAMED
NAME OF SUPERVISOR

Date : 13-MAY-2009

NOTES : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach with the letter from the organisation with period and reasons for confidentiality or restriction.

"I declare that I have read this work and in my opinion this work is adequate in terms of scope and quality for the purpose of awarding a Bachelor's Degree of Electrical Engineering (Computer)"

Signature : _____

Name of Supervisor : **Dr. Izzeldin Ibrahim Mohamed**

Date : **13 MAY 2009**

DEVOLEPMENT 5 FINGERS ROBOT HAND USING PIC

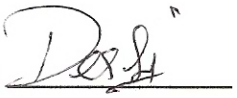
NG DER LI

A thesis submitted in partial fulfillment
of the requirements for the award of the degree of
Bachelor of Electrical Engineering (Computer)

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

MAY 2008

"I declare that this thesis entitled "Development 5 Fingers Robot Hand using PIC", is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree."

Signature : 

Name of Candidate : NG DER LI

Date : 12 MAY 2008

DEDICATION

Specially to my beloved parents, parents, siblings and friends for their eternal support, encouragement and inspiration throughout my journey of education.

ACKNOWLEDGEMENT

First would like to express my heartily gratitude to my supervisor, Dr. Izzeldin Ibrahim Mohamed for the guidance and enthusiasm given throughout the progress of this project. Under supervision, many aspects regarding this project been explored and with the knowledge, idea and support receive from him, this thesis can be presented in the time given.

My appreciation also goes to my family who has been so tolerant and supports me all these years. Thanks for their encouragement, love and emotional supports that they had given to me. Thanks to my senior and all my friends who helped me directly or indirectly in completing this project. Not forgetting, grateful appreciation is also extended to the lab technician of UTM's Laboratory who gave me great assistance during the process in accomplishing PSM I and II.

Finally, my deepest appreciation goes too my parents for their unconditionally love and support.

ABSTRACT

Robot hand is an important part of a humanoid robot. It is difficult to generate the action to emulate human hand. This thesis proposed a same-sized and light-weight robotic hand designed and concerned with the feasibility of using master glove to control 5 fingers robot hand. The 5 fingers robot hand is designed so that it can move and act like a human hand. Each finger of the robot hand has 3 degree of freedom which is almost like a human finger. The function of master glove controller is to provide the control signal to microcontroller when the manipulator's fingers move. The Servo Motor chain drive is use to drive the sector of each robot hand's finger. Furthermore, the servo motor can only provide a low torque for the finger, so robot hand only can only generate a suitable force for each finger. All the behaviour and movement of the robot are process by two PIC 18F452 microcontrollers. The master glove controller has real time control over the robot hand. So it can emulate a human hand such as grasping object. With further research and development, the robot hand can be use to implement humanoid robot.

ABSTRAK

Tangan Robot adalah satu bahagian yang sangat penting dalam satu robot kemanusian. Tesis ini berkenaan dengan reka bentuk satu tangan robot yang sama saiz serta ringan dan dapat menggunakan master sarung tangan untuk mengendalikan 5 jari tangan robot. Projek ini adalah membuat satu 5 jari tangan robot yang boleh bergerak seperti tangan manusia. Setiap jari tangan robot mempunyai 3 darjah kebebasan untuk mememulasikan seperti tangan manusia. Master sarung tangan adalah memberikan isyarak untuk mikrpengawal ketika pengguna mengubah posisi tangan. Motor yang Servo drive menggunakan rantai untuk menggerakkan sektor setiap jari. Selain daripada itu, servomotor yang hanya dapat mengeluarkan tenaga putaran rendah untuk jari tangan, jadi tangan robot hanya dapat mengeluarkan tenaga yang umum untuk setiap jari. Segala tingkah laku dan pergerakan robot di kawal oleh dua mikropengawal jenis PIC18F452. Master sarung tangan mengawal tangan robot dalam masa nyata. Jadi, tangan robot berupaya mengikut tangan orang untuk memegang dan mengerjung bahan. Dengan lebih pengajian and penyelidkan, tangan robot ini boleh digunakan dalam robot yang kemanusian.

CONTENTS

CHAPTER	SUBJECT	PAGE
	TITLE	**i**
	DECLARATION	**ii**
	DEDICATION	**iii**
	ACKNOWLEDGEMENT	**iv**
	ABSTRACT	**v**
	ABSTRAK	**vi**
	CONTENTS	**vii**
	LIST OF TABLES	**xi**
	LIST OF FIGURES	**xii**
	LIST OF ABBREVIATIONS	**xiv**
CHAPTER I	**INTRODUCTION**	1
	1.1 Background of Project	1
	1.2 Objectives	2
	1.3 Scope of Project	2
	1.4 Work Contribution	3

CHAPTER II LITERATURE REVIEW 5

2.1 Robotics and Humanoid Robot 5
2.2 Actuators 6
2.2.1 Pneumatic artificial muscles 6
2.2.2 Sensing 7
2.2.3 Servomotor 8
2.2.4 Communication 9
2.3 Existing Robot Hand 10
2.3.1 Shadow Robot Hand 10
2.3.2 Mechatronic Design of Innovative Fingers for Anthropomorphic Hands 13
2.3.3 Pinching at finger tips by humanoid robot hand 14

CHAPTER III METHODOLOGY 16

3.1 Development Process 16
3.1.1 Phase 1 – Mechanical Design Process 16
3.1.2 Phase 2 – Electronic Control Development Process 17
3.1.3 Phase 3 – Software Development Process 17
3.2 Electronic Control System 18
3.3 Project Implementation 20

CHAPTER IV HARDWARE DEVELOPMENT 23

4.1 Mechanical Design 23
4.1.1 Robot Hand Framework 23
4.1.2 String Pulling Method 27
4.1.3 The Servomotor 28
4.1.4 Master Glove Design 29

	4.2	Electronics Design	31
CHAPTER V	**SOFTWARE DEVELOPMENT**		33
	5.1	Software Design	33
	5.2	PIC Programming	34
	5.3	Pulse Width Modulation (PWM)	36
		5.3.1 Timer interrupts 0	37
		5.3.2 Timer interrupts 1	37
	5.4	MPLAB IDE	41
CHAPTER VI	**RESULT AND ANALYSIS**		42
	6.1	Motor and Finger Force Analysis	43
		6.1.1 Motor Torque and Speed	43
	6.2	Program debugging	44
	6.3	Master glove Controller	47
	6.4	Autonomous Movement	49
CHAPTER VII	**CONCLUSION AND RECOMMENDATION**		54
	7.1	Conclusion	54
	7.2	Recommendation	55
REFERENCE			56
APPENDIX A			58
APPENDIX B			61
APPENDIX C			67
APPENDIX D			85

LIST OF TABLES

TABLE NUMBER	TITLE	PAGE
2.1	The finger has 4 degree of freedom and 4 joints	10
4.1	Robot hand mechanism specification	24
4.2	Servomotor specification	28

LIST OF FIGURES

FIGURE NUMBER	TITLE	PAGE
2.1	Agonist and Antagonist	5
2.2	Connecter of the servo motor.	8
2.3	The finger of Shadow hand.	11
2.4	Shadow robot hand	12
2.5	CAD representation of the finger under development	13
2.6	Structural scheme of the endoskeleton.	13
2.7	Mechanism of Finger.	15
2.8	Inside mechanism in a palm	15
3.1	Theoretical design of the finger	17
3.2	Electronic Control Block Diagram.	18
3.3	PSM 1 Planning.	20
3.4	PSM 2 Planning.	21
4.1	Robot Hand dimension and orientation	24
4.2	Strings Tie Method for Servomotor and Finger	25
4.3	Strings Tie Method for Finger	25
4.4	First prototype built using ice-cream stick	26
4.5	Second prototype built using aluminum plate	26
4.6	Fourth prototype is a complete robot hand.	26
4.7	Third prototype consists of 4 fingers and no palm.	26
4.8	Final Robot hand consists of all the servo motors and fingers.	26
4.9	The basic idea layout of robot's finger	27
4.10	The configuration of String pulling	27

4.11	The dimension of Servomotor	28
4.12	Potentiometer rotate sensor	29
4.13	Potentiometer rotate sensor at each joint of a finger.	29
4.14	Potentiometer rotate sensor at each joint of 5 fingers.	30
4.15	Computer's power supply that used in this project	31
4.16	Schematic diagram for microcontroller	32
5.1	Robot hand program flowchart.	35
5.2	Servomotor pulse width value and its related angles.	36
5.3	Multiple PWM output generated simultaneously by PIC.	39
5.4	Flowchart for Timer Interrupt to generate PWM	40
6.1	Prototype of robot hand	42
6.2	The self made PIC Programmer	44
6.3	The MPLAB IDE programming, Simulating and debugging platform.	45
6.4	Process of one finger debugging its position locomotion	46
6.5	Master glove controller	47
6.6	Robot Hand manually control by Master Glove.	48
6.7	Robot Hand autonomous mode	49

LIST OF ABBREVIATIONS

PIC	-	Programmable Intelligent Computer
DC	-	Direct Current
LED	-	Light Emitting Diode
PCB	-	Printed Circuit Board
UART	-	Universal Asynchronous Receiver/Transmitter
PAMs	-	Pneumatic Artificial Muscles
PWM	-	Pulse-Width Modulation
PC	-	Personal Computer
CAD	-	Computer-aided design

CHAPTER I

INTRODUCTION

1.1 Background of Project

Robot hand is a part of robot arm, and it is important part for a humanoid robot. Typical applications of robot hand include welding, painting, ironing, assembly, pick and place, packaging and testing, all accomplished with high endurance, speed, and precision. It is easy to found a robot hand in now a day industrial for instead of human hand to do the dangerous job and precision job. Industrial robot hands are used in the production process and the transportation process for quality control and carrying the heavy stuff.

Robot may be used for exploration, de-mining in the military and aerospace activities. These robots use it robot hand in place of people demolition bomb to reduce casualties. Astronauts will allow the robot to check out the surrounding areas, to ensure the safety of the space. In addition, the robot hand is also widely used for helping doctor in medical surgery patients. This requires very precise technology to avoid the mistake.

Most of the robot hand in the world is making by some purpose and reason. So this project aims to investigate and development the multipurpose humanoid robot hand. The robot hand will be manipulated by using the master glove. The movement of the robot hand should be as close as possible with the human hand.

1.2 Objectives

The objectives of the project are as follows:

- The aims of this work is to develop a five fingers robot hand that is able to:
    - Pick up or handle small to medium sized objects
    - Grasping roll material
    - Perform like a humanoid robot hand
    - Control using DIY master glove controller

1.3 Scope of Project

The system consists of a mechanical design, electronic hardware, and software. The works undertaken in the project are limited to the mechanical part, electronic control, and software development.

For the mechanical part: The robot includes a finger skeleton design, motor, and caster mounting. Strings are mounted at the shaft of servo motor to pull ever sector of the finger. This robot finger is designed to operate in 3 degree of freedom. The

master glove design includes the sensor in each joint of the finger by using potential meter.

Electronic control: Choosing control system uses PIC microcontroller. Input control using potentiometer and feedback control also using potentiometer rotate sensor. The servo controller can generate the servo pulse from 0.5ms to 2.5ms when it receive the signal from main PIC microcontroller

Software development: To run a PIC, a set of C programming code have to be programmed into the device using MPLAB IDE software and modify program based on requirement of real attitude of robot hand. The phase of the robot hand is design using the Solidwork software. Software use to draw out the schematic diagram is Protel.

1.4 Work Contribution

We have developed a five fingers humanoid robot hand that has a 15 degree of freedom. Each sector of the finger was tied with 3 chain system that powered by the servo motor. The robot hand was capable to pick up small to medium size object and roll's material. The major contributions of this work are:

- We have built a robot index finger which is manipulated by the servo motor.
- We have built the robot hand other 4 fingers and successfully manipulated them by the servo motors.
- We have combined all the fingers to develop our robot humanoid hand, which was controlled and manipulated by using a PIC microcontroller.

- We have successfully controlled the robot hand by using a master glove; the output of the master glove is directly proportional to the input for the PIC microcontroller. The microcontroller generated the PWM signals which were used by the servo controller to control the shaft position of the servo motor.
- We have successfully tested our robot hand capability to pick up small and medium size objects and roll materials like ping pong ball.

CHAPTER II

LITERATURE REVIEW

2.1 Robotics and Humanoid Robot

A robot with its overall appearance based on the human body is a humanoid robot. This kid of robot have a torso with a head, two arm, two leg, one body, although some forms of humanoid robots may model only part of the body Like other mechanical robots, humanoid refer to the following basic components too: Sensing, Actuating and Planning and Control. Since they try to simulate the human structure and behaviour, most of the times humanoid robots are more complex than other kinds of robot.

Humanoid robots are used as a research tool in many areas such as military, aerospace research area, scientific areas, University development and education. Those researchers need to understand the human body structure and behavior to built and study the humanoid robots. Furthermore, the attempt to simuate the human body leads to a better understanding of it. Besides the research, developed the humanoid robots to let it can perform human tasks like personal assistance, where they should be able to assist the sick and elderly, and dirty or dangerous jobs. Regular jobs like being a receptionist or a Health services are also suitable for humanoids. In essence,

since they can use tools and operate equipment and vehicles designed for the human form, humanoids could theoretically perform any task a human being can, so long as they have the proper system.

2.2 Actuators

Actuators are the motors responsible for motion in the robot. A mechanism or robot are constructed in such a way that they mimic the animal or human body, so the actuators is to perform like a muscles and joints but with a different structure. To achieve the effect motion of the robot, robots normally use mainly rotary actuator. The common actuator is electric, pneumatic, hydraulic, piezoelectric or ultrasonic control.

2.2.1 Pneumatic artificial muscles

Pneumatic artificial muscles (PAMs) are contractile or extensional devices operated by pressurized air. Similarly to human muscles, PAMs are usually grouped in pairs (figure 2.1): one agonist and one antagonist.

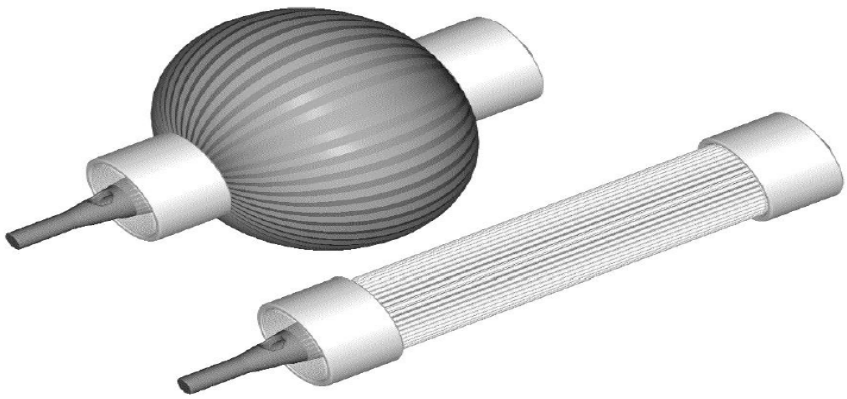


Figure 2.1: Agonist and Antagonist

PAMs are contractile and linear motion engines operated by gas pressure. Their core element is a flexible reinforced closed membrane attached at both ends to fit along each other. The mechanical power is transferred to a load. As the membrane is inflated or gas is sucked out of it, it bulges outward or is squeezed, respectively. Together with this radial expansion or contraction, the membrane contracts axially and thereby exerts a pulling force on its load. The force and motion thus generated by this type of actuator are linear and unidirectional. This contractile operation distinguishes the PAM from bellows, which extend upon inflation.

Although this type of actuator is very suitable to use as the muscle of a humanoid robot, it is hard to find in Malaysia market, so this actuator will not been used in this project.

2.2.2 Sensing

Sensor is a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument. The sensor is responsive to changes in the quantity to be measured, for example, temperature, position, or chemical concentration. The transducer converts such measurements into electrical signals which usually amplified, can be fed to instruments for the readout, recording, or control of the measured quantities. Sensors and transducers can operate at locations remote from the observer and in environments unsuitable or impractical for humans

Proprioceptive sensors sense the position, the orientation and the speed of the humanoid's body and joints. In human beings inner ears are used to maintain balance and orientation. Humanoid robots use accelerometers to measure the acceleration, from which velocity can be calculated by integration; tilt sensors to measure inclination; position sensors, that indicate the actual position of the robot or

even speed sensors. In my project, the position sensor will be mounting at every joint of the finger.

2.2.3 Servomotor

A servo motor is a generic term used for and automatic control system. The Servo is an automatic device which uses error-sensing feedback to correct the performance of a mechanism. The term correctly applies only to systems where the feedback or error-correction signals help control mechanical position or other parameters. In practical terms, that means a mechanism that you can set and forget, and which adjusts itself during continued operation through feedback.

There are numerous types of servos but they differ in their precision, speed, and strength. The connection of these servos are same, is controlled by three wires (see figure 2.2) which is negative, positive and signal. Roughly 6VDC to power the servo motor and a PWM pulse stream to indicate position.

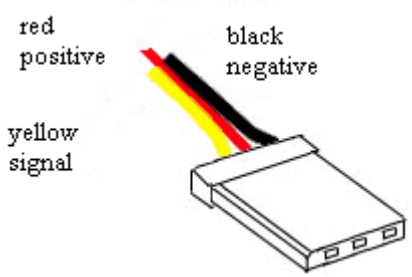


Figure 2.2: Connecter of the servo motor

A servo pulse of 1.5 ms width will set the servo to its "neutral" position, or 90°. For example a servo pulse of 1.25 ms could set the servo to 0° and a pulse of 1.75 ms could set the servo to 180°. The physical limits and timings of the servo hardware varies between brands and models, but a general servo's angular motion will travel somewhere in the range of 180° - 210° and the neutral position is almost at 1.5 ms.

2.2.4 Communication

In my project, the communication between microcontrollers and microcontroller with the computer is needed and the UART will be used.
So UART is a universal asynchronous receiver/transmitter, it is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms

By using UART, the serial transmission of digital information through a single wire or wireless medium is much more cost effective than parallel transmission through multiple wires. The transmitted information between sequential and parallel form at each end of the link can be done by using UART. Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms. Two type of Communication may be used which is full duplex and half duplex. The full duplex is both send and receives at the same time and half duplex devices take turns transmitting and receiving).

Now a day, UARTs are commonly used with RS-232 for embedded systems communications. It is useful to communicate between microcontrollers and also with PCs. Many chips provide UART functionality in silicon, and low-cost chips exist to convert logic level signals to RS-232 level signals like Maxim's MAX232 and it can be easy find at the market.

2.3 Existing Robot Hand

Robot hands have not been widely sold in the market. So this literature is based on the existing robot hand build by individual person or University research. There is only one company got sold the robot hand in this world which is Shadow Robot Company. This company manufactured a product name Shadow Robot Hand show in Figure 2.4

2.3.1 Shadow Robot Hand

The Shadow Dexterous Hand has been designed to be as similar as possible to the average hand of the human hand. The base of the forearm widens to 146mm, but the length is comparable to the human forearm. The Finger Unit reproduces as closely as possible the four degrees-of-freedom of the human finger (Table 2.3 and Figure 2.4). It has been designed to provide comparable force output and movement sensitivity to the human finger, as well as upwards-compatibility with the Shadow Dextrous Hand. Shadow Dexterous hand has 24 joints all together, with 20 degrees of freedom.

Table 2.1: The finger has 4 degree of freedom and 4 joints

Joint	Connects	Range	Muscle Type
1	Distal - Middle	-20 – +90	Coupled pair
2	Middle - Proximal	0 – +90	
3	Proximal - Knuckle	-20 – +90	Pair
4	Knuckle - Palm	-25 – +25	Single with Spring

Figure 2.3: The finger of Shadow hand

The movements of the hand are powered by a set of 40 Air Muscles in the forearm. The flow of air into and out of each muscle is controlled by eighty valves, also in the forearm. This is done based on the information gathered from the joint sensors.

The entire system is built with a combination of metals and plastics. The finger is built by using acetyl, aluminium, polycarbonate fingernails and polyurethane flesh. The finger distal can general maximum force 2.5 Nm and the finger proximal can generate maximum force 0.5 Nm.

A Hall Effect sensor measured the position with typical resolution 0.2 degrees senses the rotation of each joint. This data is sampled locally by 12-bit ADC. The sampling rate is configurable up to 180Hz. If the Tactile Sensing option is selected, then tactile sensor data is made available as per the separate Tactile Fingertip Technical Specification.

PIC18F4580 microcontroller is used for embedded control throughout the robot system. The firmware is provided as source on the host PC.

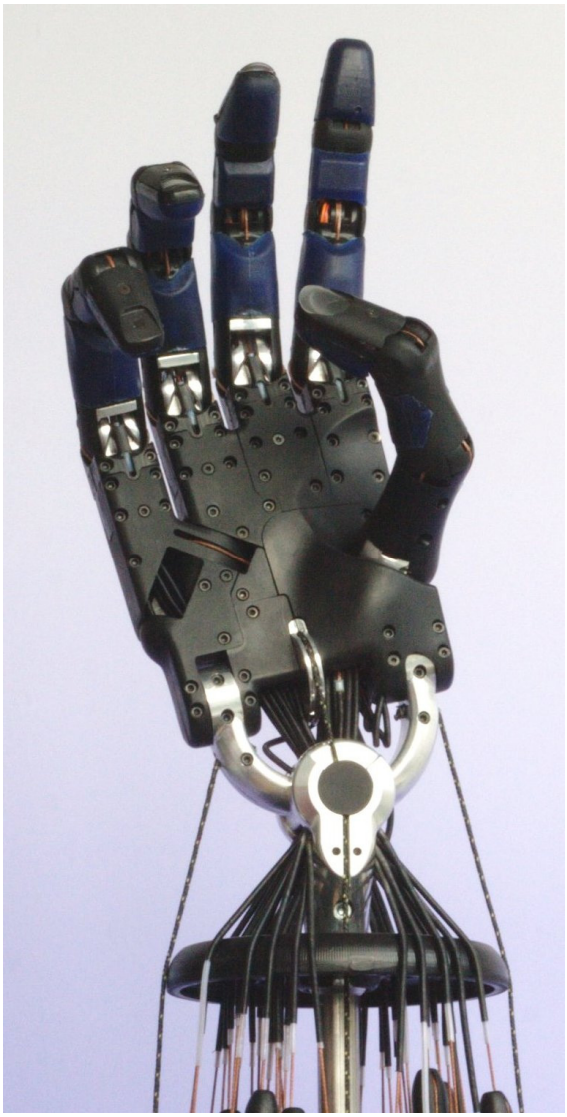


Figure 2.4: Shadow robot hand

2.3.2 Mechatronic Design of Innovative Fingers for Anthropomorphic Hands

This robot hand is development at the University of Bologna. The new design is base on the concept that the robot hand finger is explicitly addressed at the endoskeleton structure concept, so that it can host external compliant layers, like in the biological model of the human hand, in order to increase contact adaptability and grasp robustness and stability (Figure 2.5)

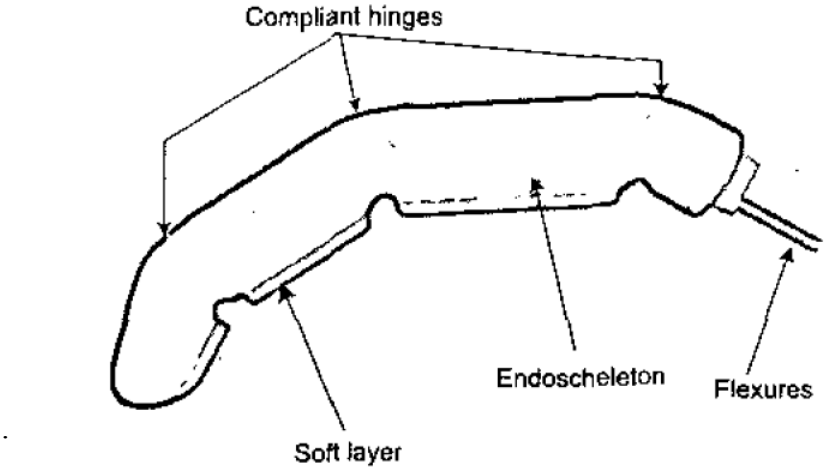


Figure 2.5: CAD representation of the finger under development

The actuation of the finger is provided by remote linear actuators like at currently linear synchronous motors with the motion transmission obtained with flexible elements routed with low-friction linear guides, no any pulley or other non-biomorphic devices are being used.

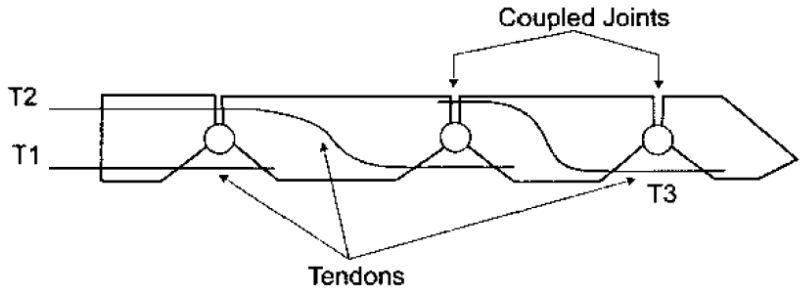


Figure 2.6: Structural scheme of the endoskeleton.

The structural scheme of the endoskeleton show in the Figure 2.6 got only three parallel joints have been implemented, and the adduction-abduction joint is not present. The proximal and the medial joints are independently actuated, while the distal joint is coupled to the movement of the medium joint. Joint actuation is powered by remote motors. The material used to built this finger is high strength steel.

There are several sensor been used in this project, that is position sensor and force sensor. The positions sensor is links of the finger, a measure based on tendon lengths and the normally the sensor is potentiometers and hall- effect based sensors. Strain gauge sensor and tactile sensors was distributed under the soft skin of the finger.

2.3.3 Pinching at finger tips by humanoid robot hand

This project is design by the Kiyoshi Hoshino which under the Institute of Engineering Mechanics and Systems at University of Tsukuba. The robot hand design to stably pinching paper or needle with the finger tips. The authors firstly propose a small-sized and light-weight robotic hand and the secondly propose a new robot hand capable of properly realizing a pinching motion with finger tips. Author also focuses on additions degree of freedom of twisting motion to the thumb. Author design the robot with few characteristic:

- The robot hand size and shape close to those of the humans hand.
- Degree of freedom motion to be furnished on the robot should be sacrificed to some extent.
- The motor and reduction gears can be incorporated in the hand

From the figure 2.7 show a motor with encoder and reduction gear occupies mechanism space of the robotic hand with most non-compromising manner. Four

fingers except for the thumb have three joints referred to as MP joint, PIP joint, and DIP joint.

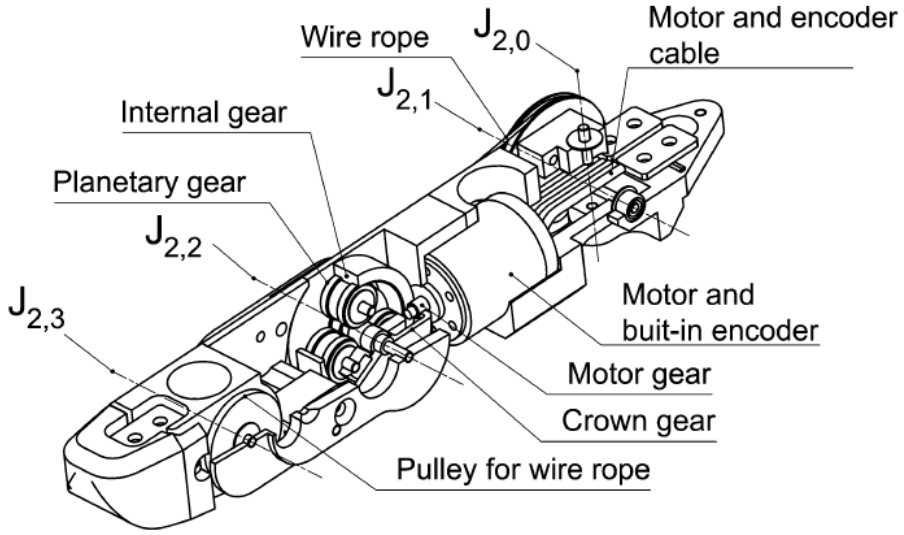


Figure 2.7: Mechanism of Finger.

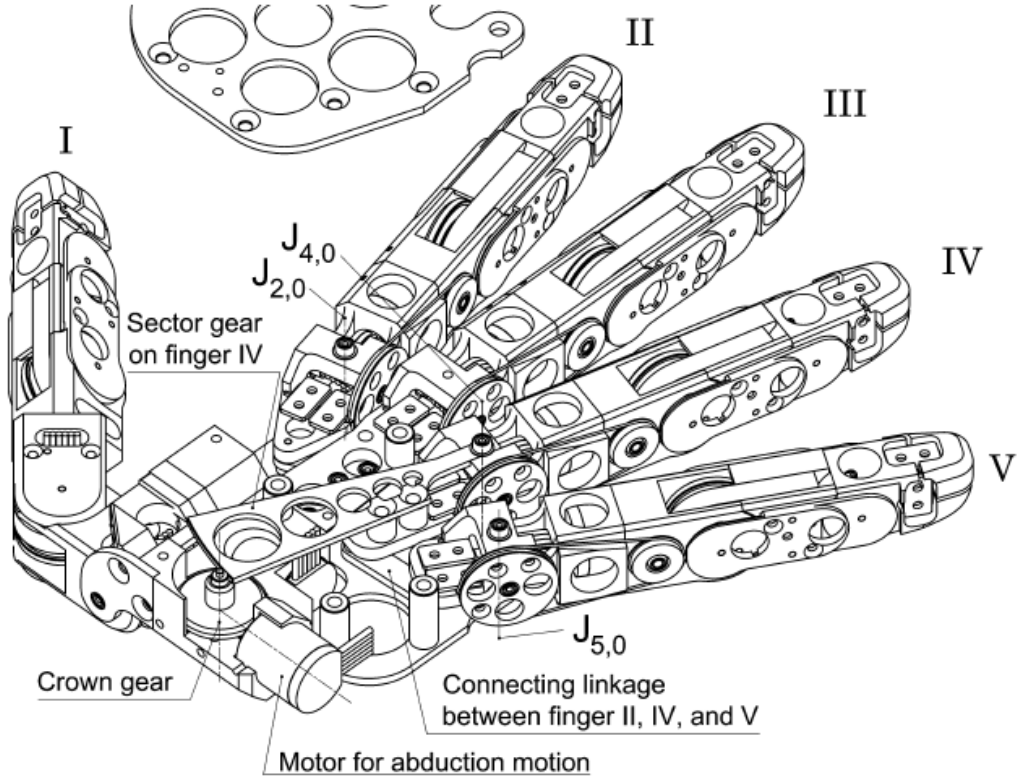


Figure 2.8: Inside mechanism in a palm

CHAPTER III

METHODOLOGY

3.1 Development Process

The objective of this project is to develop and investigate the feasibility of using master glove to manipulate a 5 finger robot hand. To do so, the methods and technical strategies implied is the most important disciplined need to look at. Therefore some simplified phase by phase method was proposed. By using this method, problems can be detected at the early stages to avoid hectic failures. There is always a target to reach either short term or long term goals. It is more organized to do the job one by one according to their respective phases. Thus, the development of the robot is divided into three phases. They include; mechanical design, electronic control system and software development process.

3.1.1 Phase 1 – Mechanical Design Process

Mechanical design is one of the major phases in the development of the robot hand. This part contributes to what the robot hand would look like. The skeleton of the robot hand is designed and constructed in this phase. The purpose of the skeleton

is to provide a place to mount the electronics component such ask sensor, and servo motor for the robot finger. Theoretical design for the robot finger is done by using the servo motor to pull each sector of the finger (Figure 3.1)

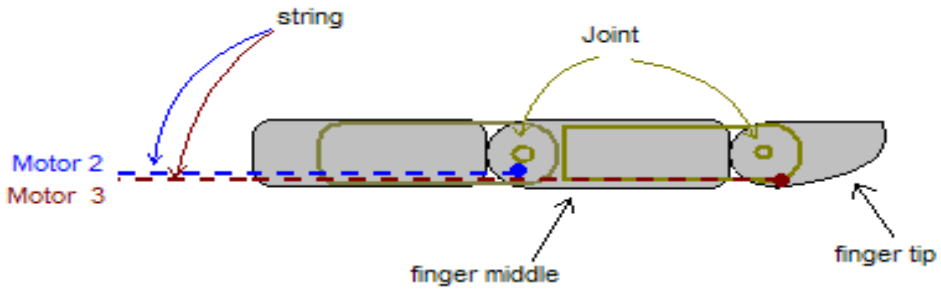


Figure 3.1: Theoretical design of the finger

3.1.2 Phase 2 – Electronic Control Development Process

Electronic Control Development Process is the most complex phases as it covers many tasks which all need specific attention. The second phase of the robot development involves system control circuit, sensor interface circuit and servo motor control circuit. Subsequently, numerous tests on the designed circuit are performed on a prototyping board. Once the circuit works effectively, the circuit designed then is transferred on to a dot nut board. There are several intermediate steps, which comprises of drawing a schematic diagram and PCB layout using commercial software such as Portel.

3.1.3 Phase 3 – Software Development Process

The first step in this stage is to select an appropriate type of microcontroller, or in other words to recognize the appropriate PIC microcontroller language for the robot's programming through MPLAB IDE software. Programming is an art of making the robot better and smarter. The C18 Student Edition provided by the company Microchip is a compiler for the MPLAB IDE. The setup of such

microcontroller system involves interfacing with the sensor and motor control circuit.

To design and draw the mechanical part of the robot hand, a software call Solidwork is needed. This software can let us drawing a 3D diagram and can change the 3D layout to 2D layout.

3.2 Electronic Control System

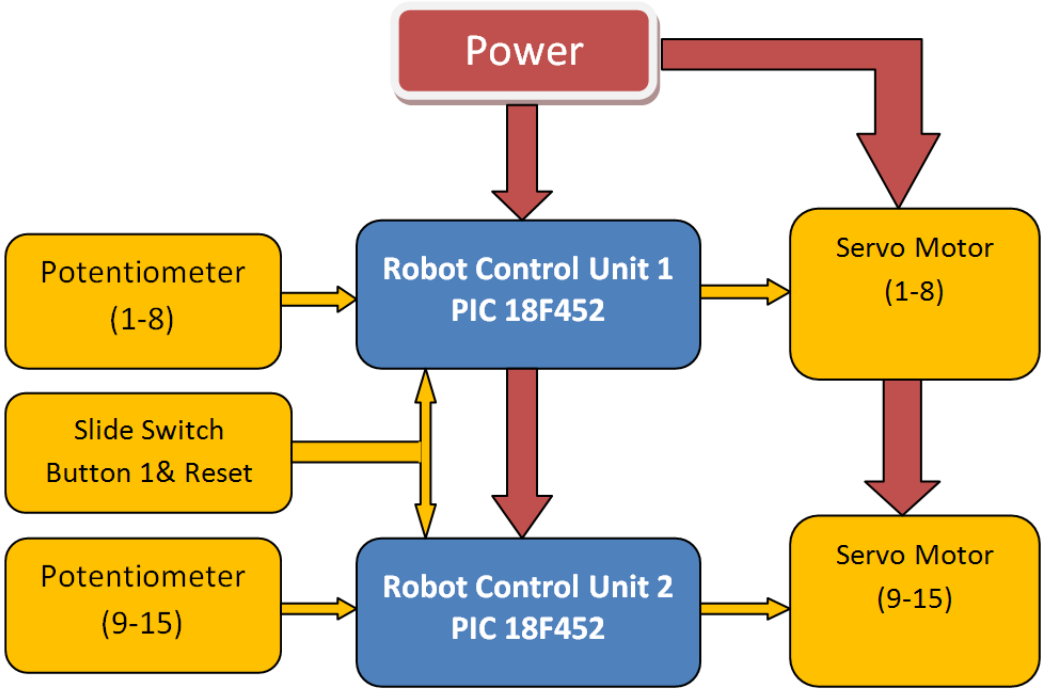


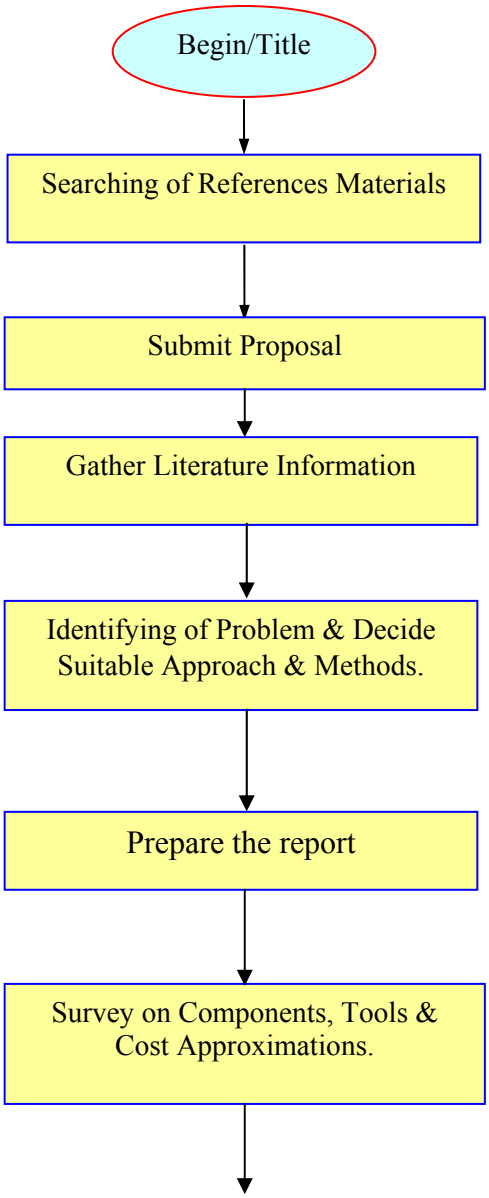
Figure 3.2: Electronic Control Block Diagram

The block diagram in Figure 3.2 shows how the electronic control system was developed and how they communicate with each other. There are 8 blocks at the block diagram show the Power, Input, Microcontroller center unit 1, Microcontroller center unit 2 and Output. There have 2 microcontrollers PIC18F452 to control servomotor. The input of this system have a reset button, button 1, slide switch and 16 signal from the potentiometer at the master glove. The outputs generate by the

microcontroller are PWM that used to control the servomotors. This system is an open loop system, because there has no feedback from the robot hand.

3.3 Project Implementation

In development stages, planning is the most important aspect. Proper planning is important to make sure that the project will be constructed successfully in time. Figure3.3 and 3.4 shows the project's planning for PSM1 and PSM2.


Begin/Title
Searching of References Materials
Submit Proposal
Gather Literature Information
Identifying of Problem & Decide Suitable Approach & Methods.
Prepare the report
Survey on Components, Tools & Cost Approximations.

(To PSM 2)

Figure 3.3: PSM1 Planning

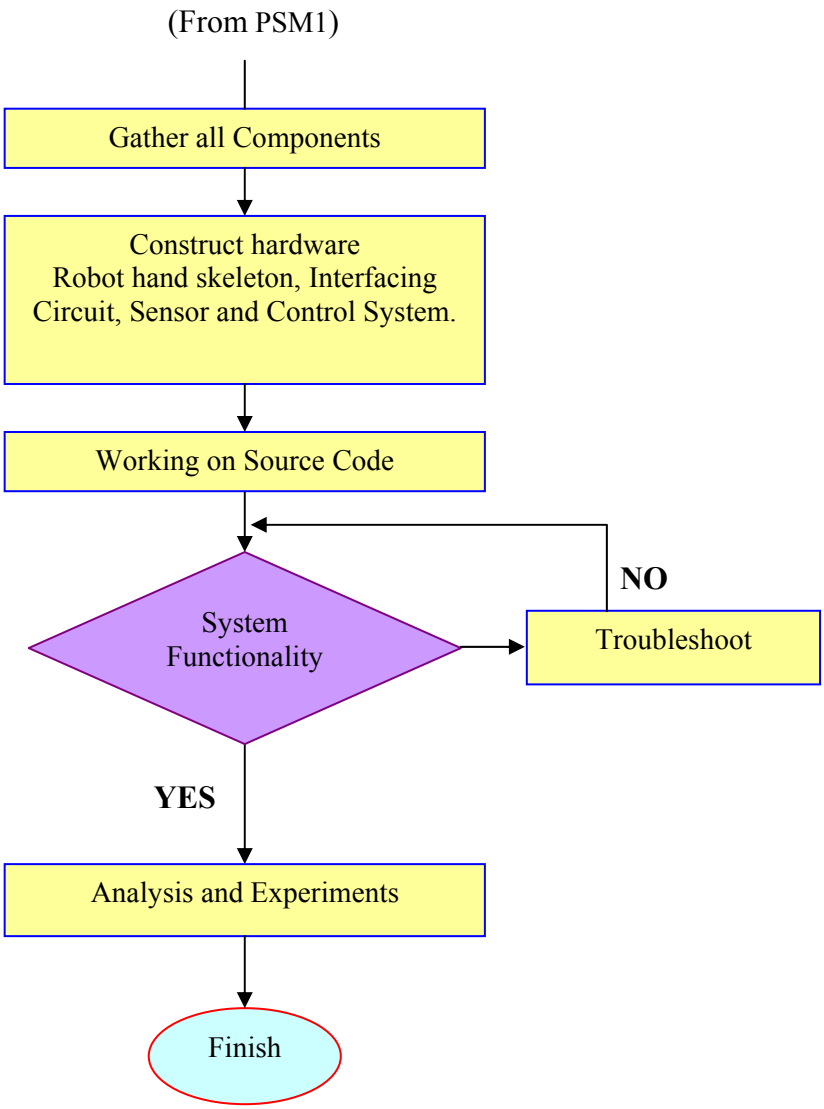


Figure 3.4: PSM 2 Planning

The elements of the Figure 3.3and 3.4 explain as the step planning for PSM. For the part searching of reference materials explain that searching and reference all the materials are obtained from published journals, books and internet relative about my PSM project. Then, submit proposal. A proposal is prepared for supervisor. At the gather literature information part obtained information will be evaluated and the appropriate portion or parts of the literature will be adopted into the project. It will be as references or guidance in developing the project. After that is identifying of problem, deciding suitable approach and methods. Once the literature review was

finished, next is methodology will used to identified the problem and decide the suitable approach and method in solving the problem. After finish all the task above, a report for PSM1 will be prepared. Next, a survey on components, tools and cost approximations will be carried out. Once the development system has been decided, a survey on the components and tools and the cost approximation will be done so that the budget in constructing the robot is not too expensive.

For the planning PSM 2, first at all, all the components will be gathered followed by the construction stage or hardware development. It will only start after all the components and tools are available. It includes building of Robot hand skeleton, interfacing circuits and the controller. After that, the controller used must be programmed to make sure it worked properly. The language used to program the controller is a programming C language through MPLAB IDE software. After complete all the construction, the functionality of the system will be tested. If there is a problem troubleshoot will be performs.

CHAPTER IV

HARDWARE DEVELOPMENT

This chapter presents the framework of the 5 fingers robot hand. The hardware can be divided into mechanical and electronic design. All the components and techniques applied in this project are also presented in this chapter.

4.1 Mechanical Design

4.1.1 Robot Hand Framework

The chassis specifications were summarised in Table 4.1 while the dimensions and orientations for the chassis were shown in Figure 4.1.

Table 4.1: Robot hand mechanism specification

Design Factor	Specification	Description
Weight	Total = 1.2 kg	The weight includes the forearm and all the servomotors. As long as the overall robot hand weight is not heavier than human hand then it is acceptable.
Size	300mm x 245mm x 150mm (length x width x high)	The size of the framework is according to the size of human hand.
Material	Frame : Aluminum plate	Material are relatively inexpensive, light and available everywhere.

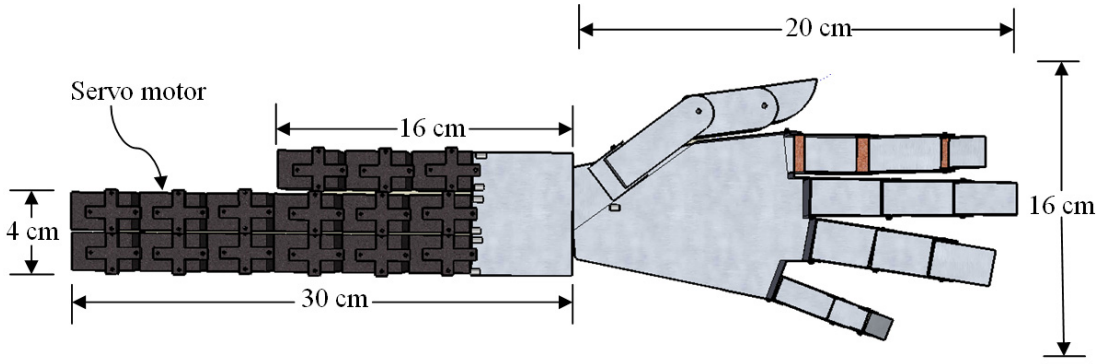


Figure 4.1: Robot Hand dimension and orientation

The Robot hand was constructed based on the human hand's shape with 3 joints at each fingers. There were 3 degrees of freedom for each finger which made those fingers can bend 90^{o} at every finger sector. The sixteen servo motors were placed at the forearm which acted as the muscle of the robot hand's fingers. Strings were tied from the shaft of servomotor to each sector of finger to mimic the tendon and muscle of the fingers. The upper string was tied on the left side of the shaft and the lower string was tied on the right side of the servomotor's shaft.

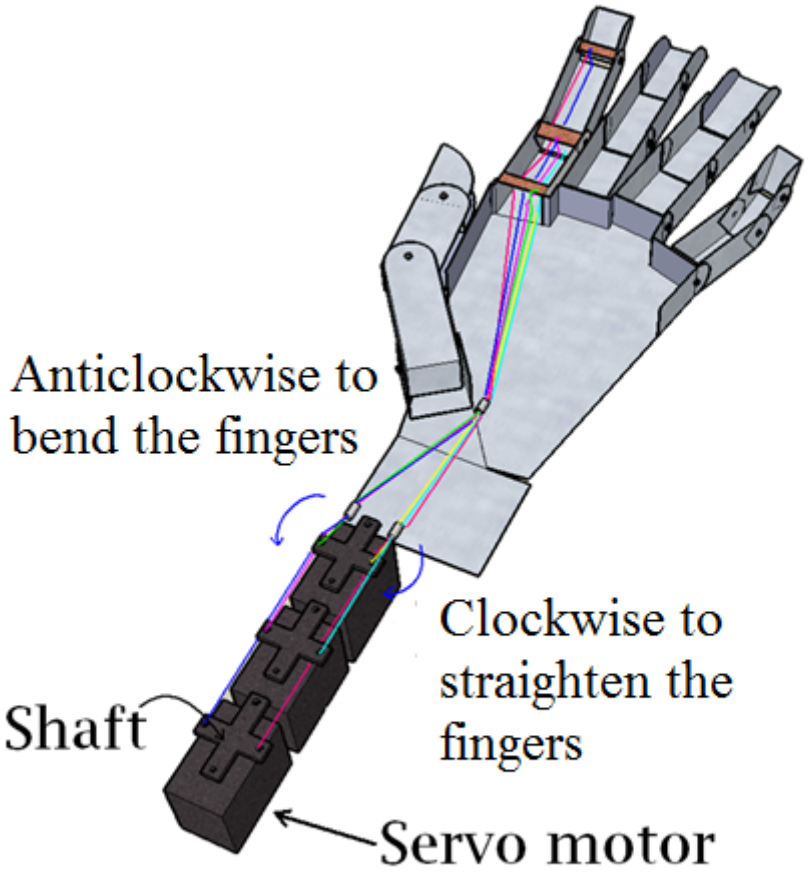


Figure 4.2: Strings Tie Method for Servomotor and Finger

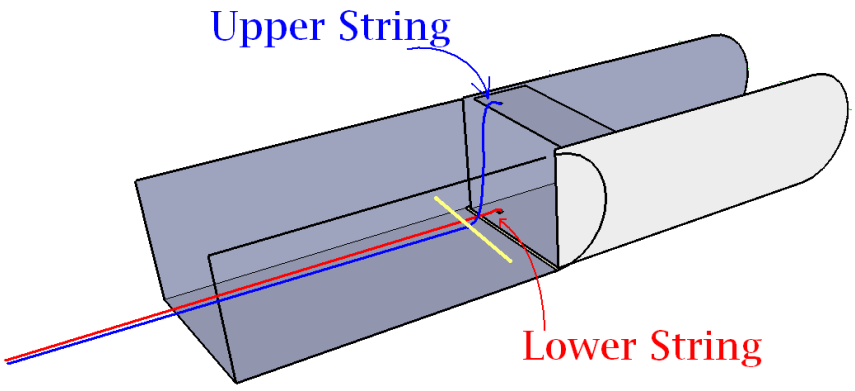


Figure 4.3: Strings Tie Method for Finger

The illustration below summarized the Robot hand structure and mechanism design process of the revolution in the Robot hand prototype.

Figure 4.4: First prototype built using ice-cream stick

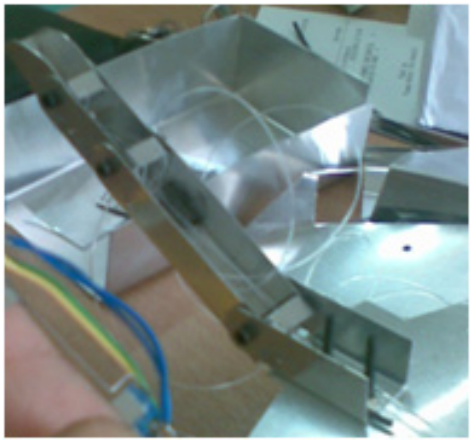


Figure 4.5: Second prototype built using aluminum plate

Figure 4.6: Fourth prototype is a complete robot hand.

Figure 4.7: Third prototype consists of 4 fingers and no palm.

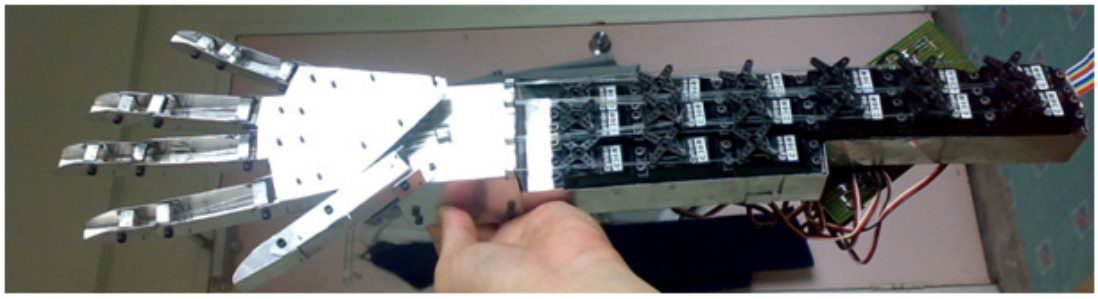


Figure 4.8: Final Robot hand consists of all the servo motors and fingers.

4.1.2 String Pulling Method

This robot hand's fingers were moved by controlling the servo motor on the forearm of the robot hand. The strings tied from the servomotor and the finger's sector can pull those finger sectors to bend up to 90°. The servomotor's shaft turned anticlockwise by 90° to make the finger bends upward 90° and if servomotor's shaft turned clockwise by 90° , it will make the finger go back to the straight position. Configuration diagram of this string pulling method was shown in Figure 4.10.

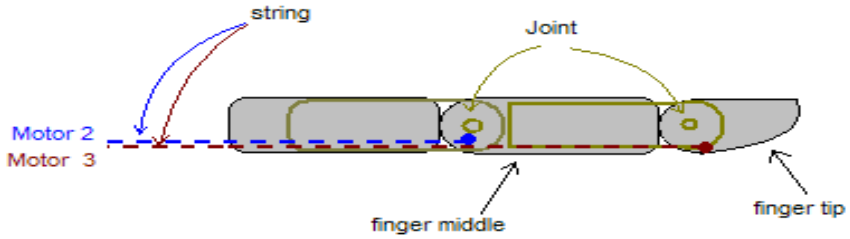


Figure 4.9: The basic idea layout of robot's finger

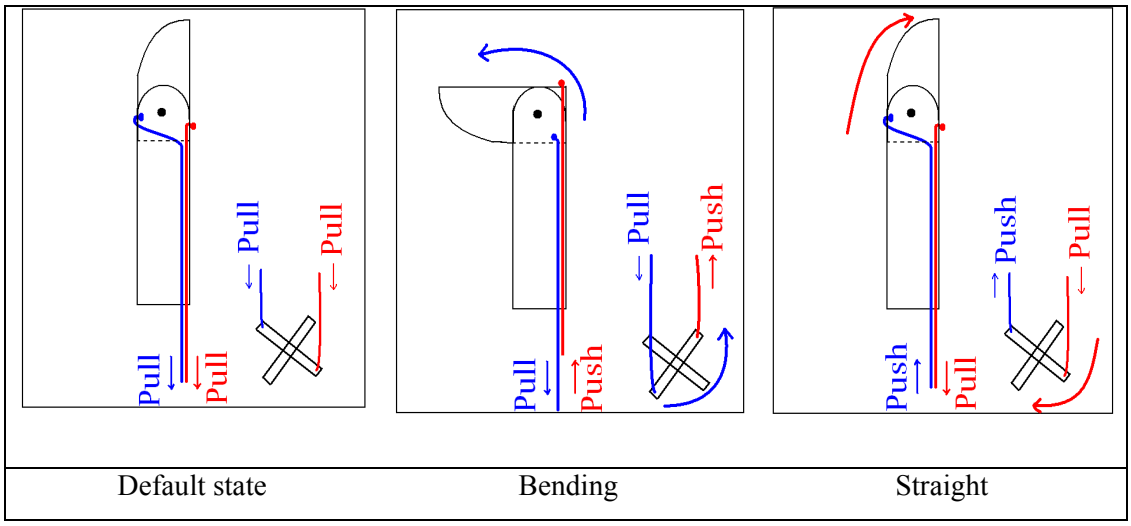


Figure 4.10: The Configuration of The String Pulling

4.1.3 The Servomotor

A servo motor (servo) is an electromechanical device in which an electrical input determines the position of the armature of a motor. It was used to power this robot hand because its torque is felicitous and the position can be determined by using PWM. Figure 4.11 showed the dimensions of the servomotor.

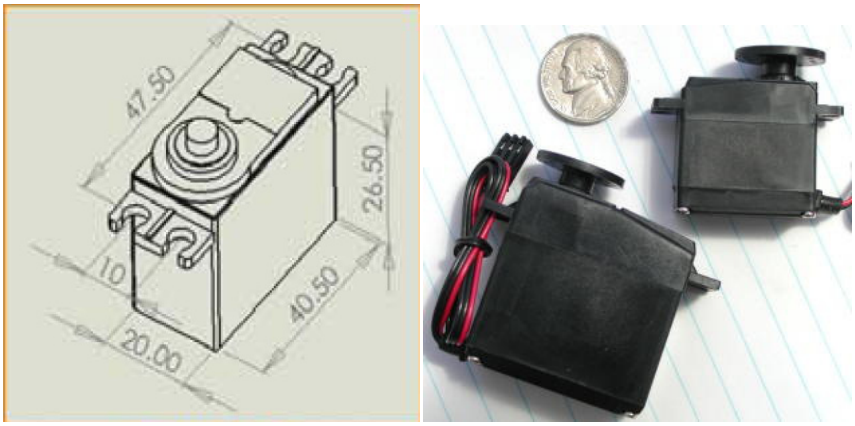


Figure 4.11: The dimension of Servomotor

The specifications of the Servomotor were listed as below:

Table 4.2: Servomotor specifications

Parameter	Specification
Supply	**: 5V**
Speed	**: 0.14 s/60°**
Torque	**: 4.50 kg.cm**

4.1.4 Master Glove Design

A potentiometer is a three-terminal resistor with a sliding contact that forms an adjustable voltage divider. Potentiometers are commonly used to control electrical devices such as a volume control of a radio. Potentiometers which are operated by a mechanism can be used as position transducers, for example, in a joystick. However, in this project, the potentiometer will be constructed at every joint of the master glove to measure the position of our hands. The output voltage from the potentiometers which was converted to 10 bits digital value will be saved in the ADC register of the microcontroller. The potentiometer was shown in Figure 4.12 and the one finger's master glove controller was shown in figure 4.13.

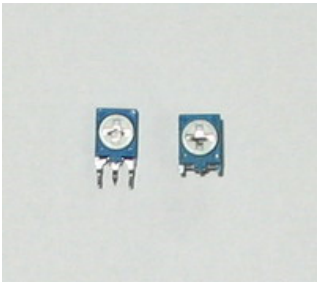


Figure 4.12: Potentiometer rotate sensor

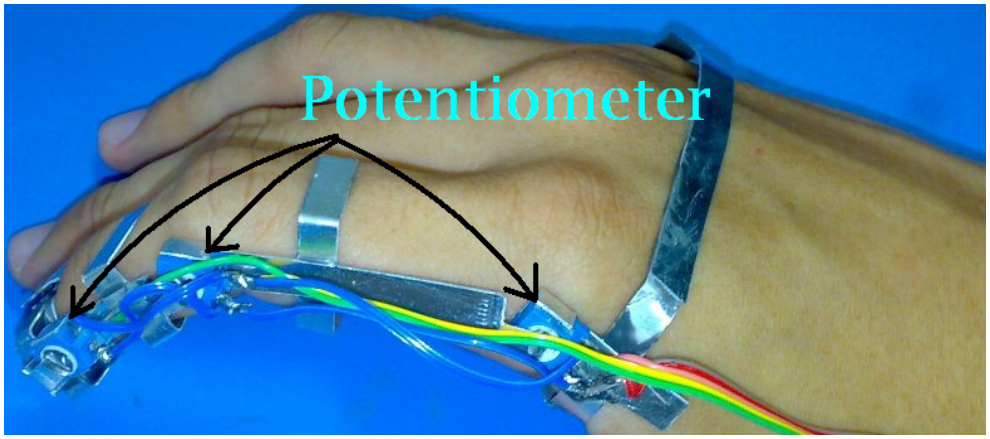


Figure 4.13: Potentiometer rotate sensor at each joint of a finger.

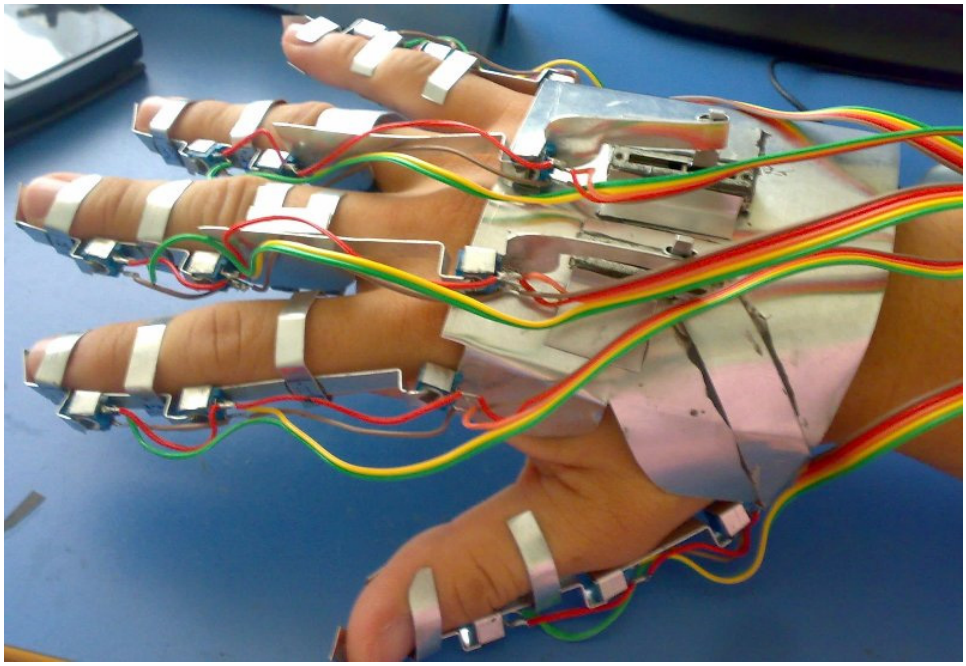


Figure 4.14: Potentiometer rotate sensor at each joint of 5 fingers.

4.2 Electronics Design

The microcontroller used in this project to control the robot hand is PIC18F452. It is the brain for the robot where it controls all the robot behaviors. The microcontrollers are easy to be used. It can be used to interface with motors, produce a variety of displays as output devices, communicate to PCs, read external sensor values and even connect to a network of similar controllers as well as to do all of these tasks without many extra components. This leads to a small and compact system that is more reliable and cost-effective. All the potentiometers will be the input for the microcontroller while the output from the microcontroller PWM was connected to the Servomotor. Figure 4.16 showed the main layout circuit for the electronic control system. It needed two PIC18F452, two 20MHz crystal and 5V supply. The oscillator circuit is used to provide an accurate and stable periodic clock signal to PIC18F452 microcontroller. The range of clock frequency could be changed from 4MHz to 25MHz. The clock frequency will determine the speed of the microcontroller executing the instructions. There is no voltage regulator in the layout circuit because the 5V supply is directly connected to the Computer power supply. The figure 4.15 shows computer's power supply that was used in this project.

The microcontroller needs to be programmed so that it can perform the predetermined tasks. The software used to program the PIC18F452 microcontroller is the MPLAB IDE software from Microchip Inc.

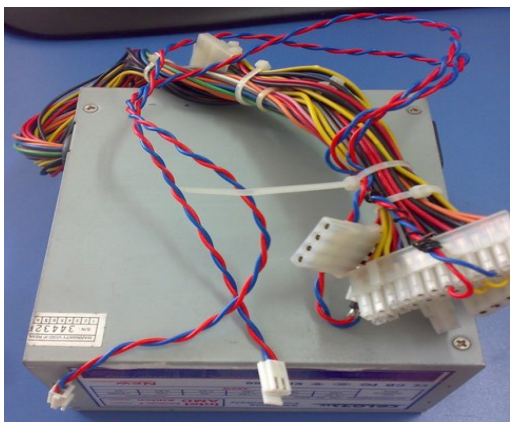


Figure 4.15: Computer's power supply that used in this project

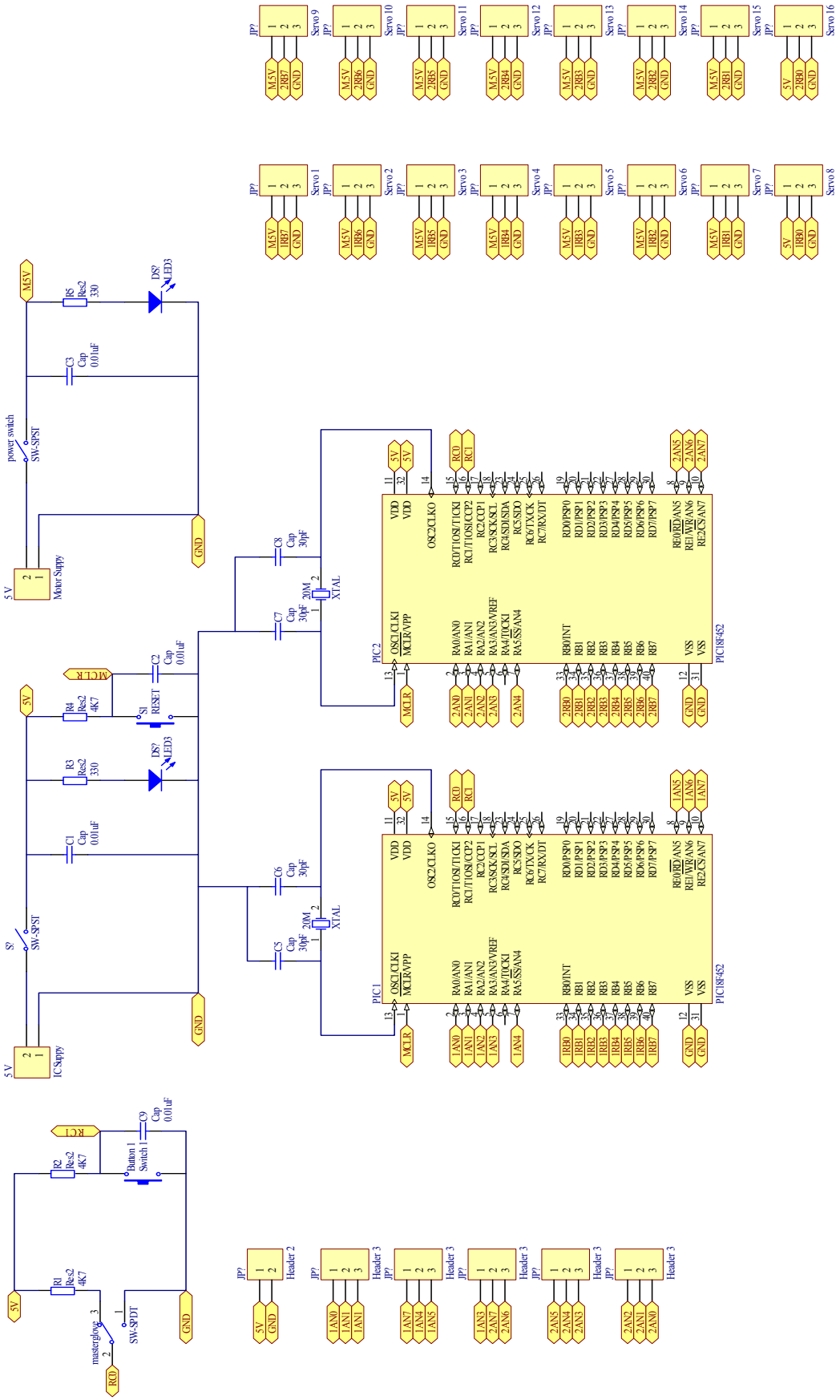


Figure 4.16: Schematic diagram for microcontroller

CHAPTER V

SOFTWARE DEVELOPMENT

5.1 Software Design

Apart from developing the hardware, software is needed for the project to function. This software is needed by the PIC microcontroller to take the appropriate action to produce an output from the input that it senses. This software can be developed using C++, MPLAB IDE, or PIC Basic. In this project, MPLAB IDE was used to produce the software, where programming C language was applied to write the program and the compiler C18 was used to compile the program.

Besides the programming C, there are other types of programming language that can be used to program the PIC microcontroller. This includes the high level language, the assembly language, and the low level language (machine language). Each of this software has their own advantages and disadvantages. Programming C language is used to write the program, because this programming C can be used to represent the machine language instruction with alphanumeric characters and it is easier to be written.

5.2 PIC Programming

The robot hand programming flow chart will be shown on this part and will be converted into the C-language together with the pic18f452.h library using the student version of the C-18 PIC complier. The robot hand program flow chart is shown below in figure 5.1. The full program is attached on the appendix.

From the flowchart, there are two modes in the program to control and manipulate the robot hand. The first mode is autonomous mode, which was pre-programmed for the robot hand to move by following the sequence of instruction. The second mode is manually controlled by the master glove. This program is about to read the data from the analog input and change it to timer interrupt to generate PWM. If both the modes are not activating, all the fingers will be in the default position, which those fingers will be in straight position.

There are two parts of the program which are the main program and the interrupt program. The microcontroller will always run the main program until there is an interrupt occurred. When microcontroller receives an interrupt flag, it will jump to interrupt process.

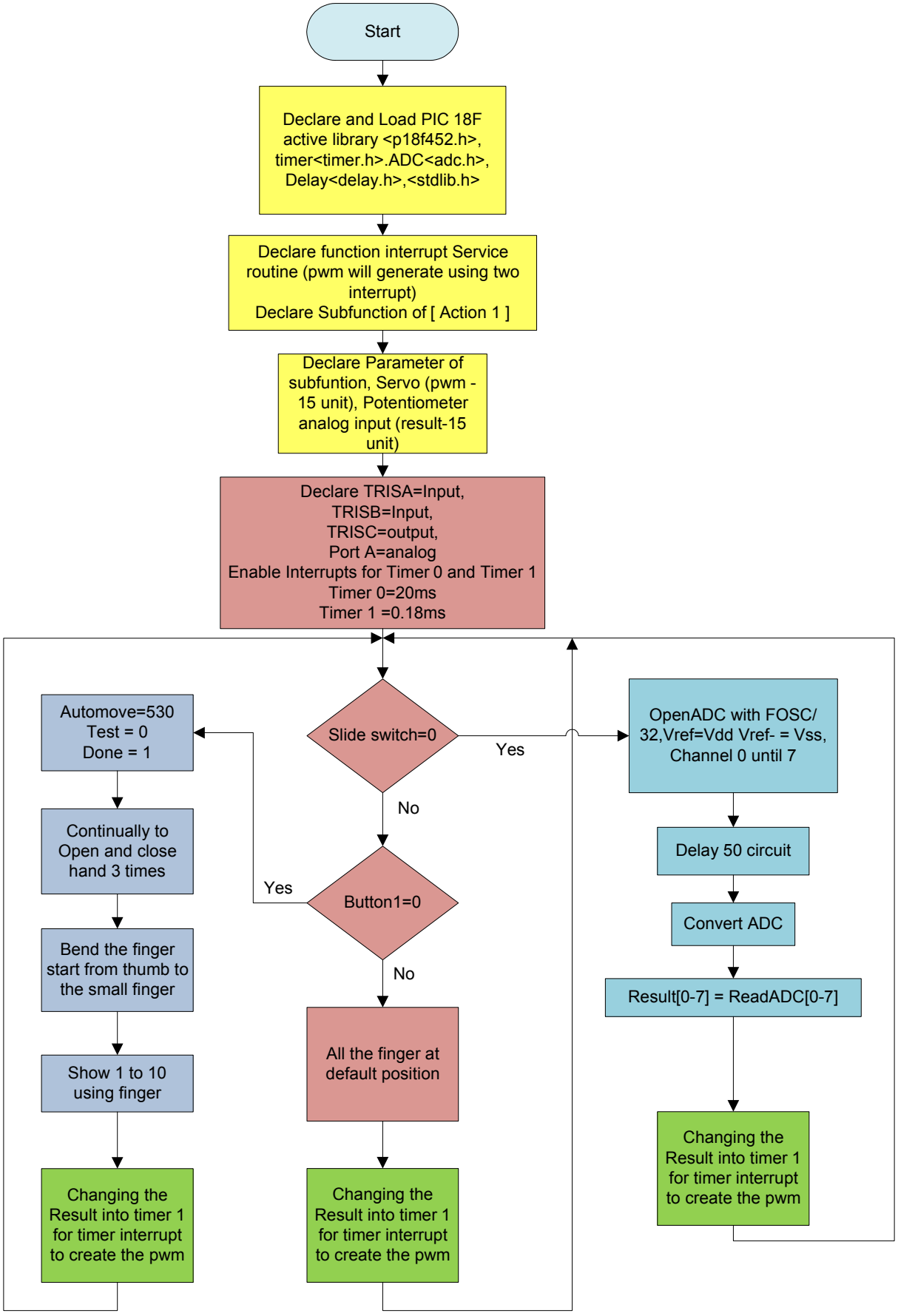

Start
Declare and Load PIC 18F active library <p18f452.h>, timer<timer.h>.ADC<adc.h>, Delay<delay.h>,<stdlib.h>
Declare function interrupt Service routine (pwm will generate using two interrupt)
Declare Subfunction of [Action 1]
Declare Parameter of subfuntion, Servo (pwm - 15 unit), Potentiometer analog input (result-15 unit)
Declare TRISA=Input,
TRISB=Input,
TRISC=output,
Port A=analog
Enable Interrupts for Timer 0 and Timer 1
Timer 0=20ms
Timer 1 =0.18ms
Slide switch=0
Yes
No
OpenADC with FOSC/ 32,Vref=Vdd Vref- = Vss, Channel 0 until 7
Delay 50 circuit
Convert ADC
Result[0-7] = ReadADC[0-7]
Changing the Result into timer 1 for timer interrupt to create the pwm
Button1=0
Yes
No
Automove=530
Test = 0
Done = 1
Continually to Open and close hand 3 times
Bend the finger start from thumb to the small finger
Show 1 to 10 using finger
Changing the Result into timer 1 for timer interrupt to create the pwm
All the finger at default position
Changing the Result into timer 1 for timer interrupt to create the pwm

Figure 5.1: Robot hand program flowchart.

5.3 Pulse Width Modulation (PWM)

Pulse-width modulation (PWM) of a signal or power source involves the modulation of its duty cycle, to either convey information over a communications channel or control the amount of power sent to a load. The PWM is used to control the position of the servomotor. The example of the calculation of servo motor angle responds to the pulse width modulation used throughout this robot hand program is shown below, including a graphical illustration of the servo position related to its pulse width value. Two timer interrupt was used to generate the PWM for controlling the servomotor position which included timer interrupt 0 and timer interrupt 1.

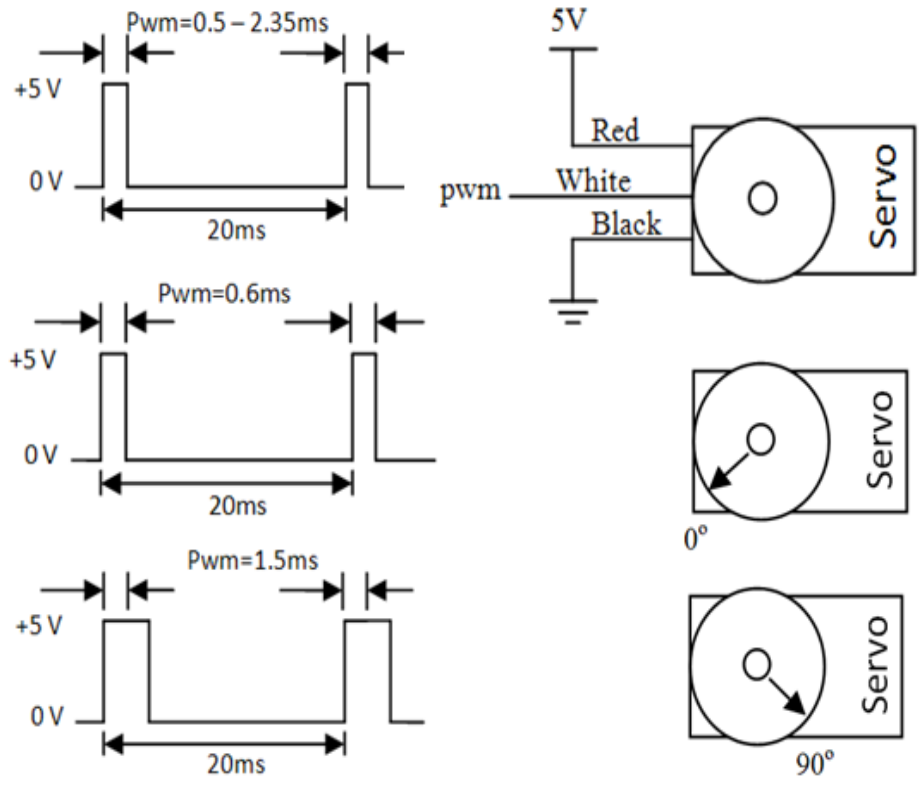


Figure 5.2: Servomotor pulse width value and its related angles.

5.3.1 Timer interrupt 0

The pulse width generating program and its calculation are shown here where the 18F452 PIC microcontroller, is 16 bits, thus there is 2^{16} = 65536 value can be generated, and the crystal clock for the circuit is 20MHz, and the perscale factor is 2, thus the PWM period calculation are shown below:

$$Timer0 = 65536 - \frac{pwm(\mu s)}{4 \times 0.05 \times 2} \quad \text{..... (5.3.1)}$$

Maximum timer value = 65536
Period for PWM = 20 ms
Substitute PWM to equation (5.3.1); Timer0 = 15536
Timer0 = $3CB0

Hence, the timer interrupt 0 will be set to value $3CB0 and when it flows from $FFFF to $0000, it will jump to the interrupt subroutine program.

5.3.2 Timer interrupt 1

The timer interrupt 1 is used to generate the pulse of the PWM. The timer1 also is in 16 bits, thus there is 2^{16} = 65536 value can be generated, but the prescale factor is 1, thus the pulse width calculation show below:

$$Timer0 = 65536 - \frac{pwm(\mu s)}{4 \times 0.05 \times 1} \quad \text{..... (5.3.2)}$$

Analog input range = 2.53 to 0.5 V
Voltage reference, Vref = 5V
10bits in ADC register = 2^{10} = 1024

Analog voltages convert to Digital value

$Result = \frac{V \times 1024}{5}$…….. (5.3.3)

Maximum digital voltage value: $\frac{2.53 \times 1024}{5}$ =527

Minimum digital voltage value: $\frac{0.5 \text{ x } 1024}{5}$ =102

Range between digital voltage values: 527-102=425

Maximum Timer 1 = 62535

Minimum timer 1 = 58035

PWM timer1 range; 62535 - 58035 = 4500

Formula for converting Digital voltage value to interrupt timer1 to create the PWM was generated by using the range of PWM and the range of digital voltage values.

$$Sv01 = \frac{result}{425} \times 4500 + 56955 \text{ ……..} \quad (5.1)$$

The value of timer0 and timer1 was determined at the beginning of the main program, so that the interrupt can happen when the program starts to run. The timer interrupt 0 will make the interrupt happen at every 20 ms and the timer interrupt 1 will make the interrupt happen depending on the pulse width of each PWM output. Using this method, the output of the PWM has been generated simultaneously. The output of the PWM and flowchart of the timer interrupt were shown in figure 5.3 and figure 5.4

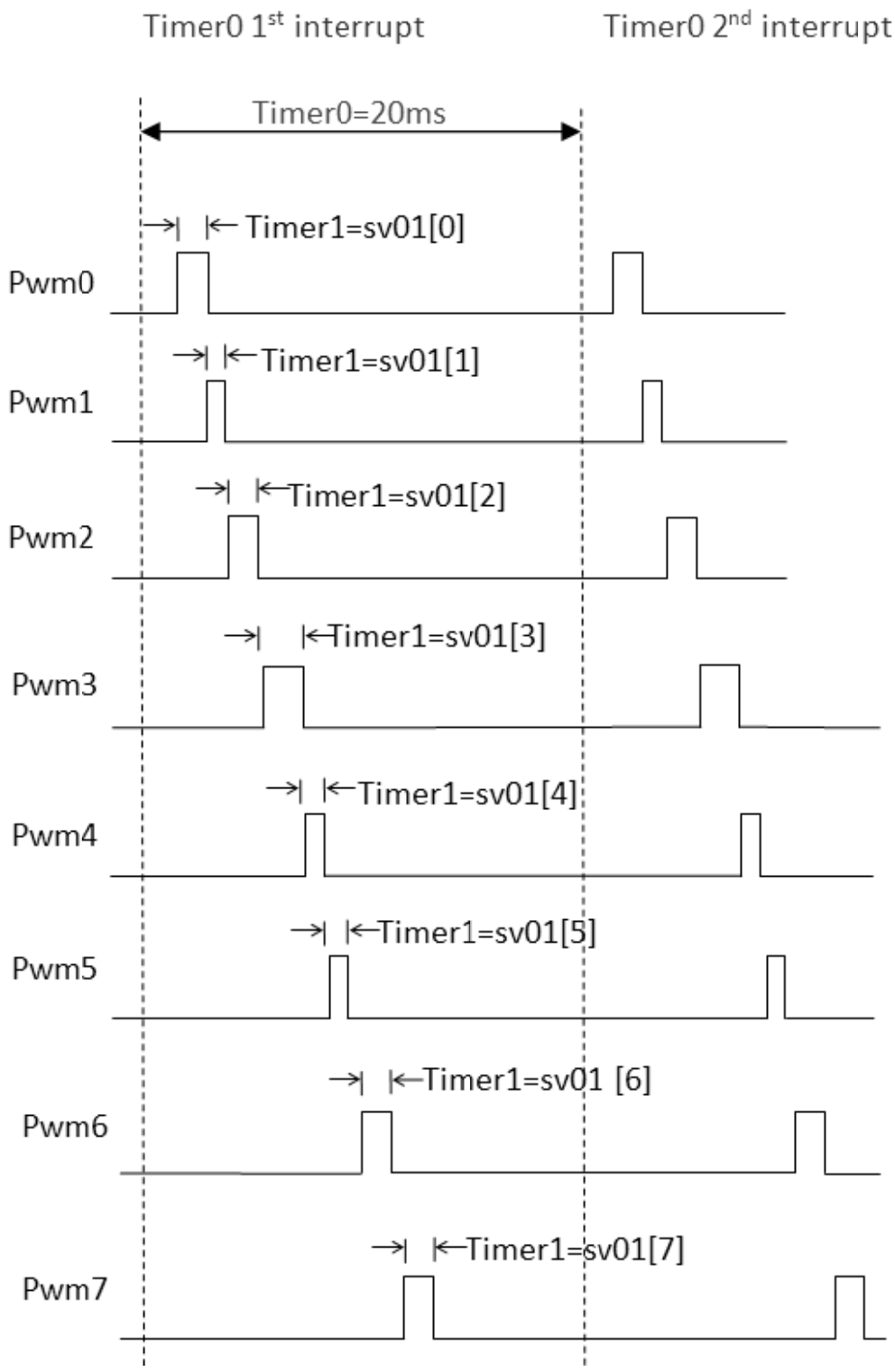


Figure 5.3: Multiple PWM output generated simultaneously by PIC.

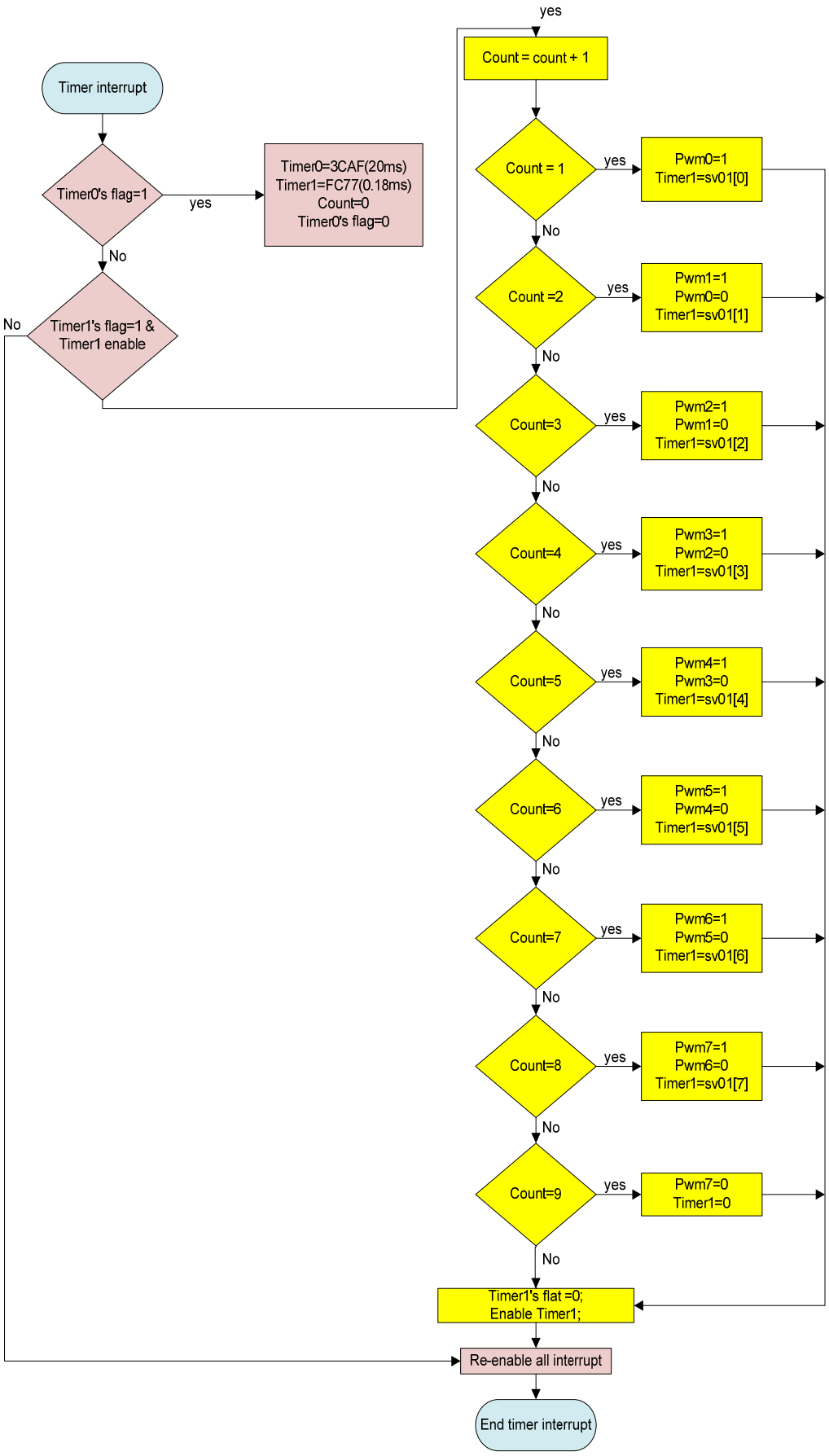


Figure 5.4: Flowchart for Timer Interrupt to generate PWM

5.4 MPLAB IDE

In order to program the PIC microcontroller, MPLAB IDE software is used where it offers a project manager and program text editor besides the user-configurable toolbar that containing four predefined sets and a status bar which communicates editing and debugging information. The programming of PIC microcontroller is achieved through the assembly language. The advantage of this software is that it can write, debug and optimize the PIC microcontroller based application.

CHAPTER VI

RESULT AND ANALYSIS

This chapter discussed on the outcome of the project where analysis and experiments are being conducted in order to test the functionality and performances of the robot hand. The complete prototype of the robot hand shows in Figure 6.1.

Figure 6.1: Prototype of robot hand

6.1 Motor and Finger Force Analysis

6.1.1 Motor Torque and Speed

The speed of the servo motor depended on the changes of analog input of the master glove; it was directly proportional to the voltage change of the potentiometer. However, it had two maximum speeds for the servo motor. The power supply for this robot hand's servomotor was 5V for this project, so the maximum speed was 0.14 s/60° and the fingers maximum speed was also 0.14 s/60°. Technical analysis on torque of the robot hand's fingers was calculated and the length of the shaft and the length of the finger were measured. The torque could be predicted by;

Torque of Servomotor= 4.50 kg.cm

Radius of the Shaft, r = 1.20 cm

Force of the string, $F_s = \frac{\tau}{r} = \frac{4.5}{1.2}$

= 3.75 kg.cm/s²

Radius of the joint, r_j = 0.70 cm

Torque of the Fingers = F_s x r_j

= 3.75 x 0.7

= 2.625 kg.cm

Hence, the torque was reduced from 4.50 kg.cm to 2.625 kg.cm.

6.2 Program debugging

After the robot hand hardware, structure, electronic system and basic robot hand locomotion program were tested, the next stage was the programming enhancement of the robot where interesting pattern and movement will be tested to fully optimize the 15 degree of freedom of the robot hand. In this stage, creativity and observation of human hand movement as well as characteristics play important roles to enhance the robot hand movement. The aid of support circuit of DIY PIC Programmer plays an importance role to debug the program and it is used to flash program into the microcontroller. The MPLAB IDE software is also able to support the debugging process by using the MPLAB SIM to simulate and debug the process until it is fully utilized.

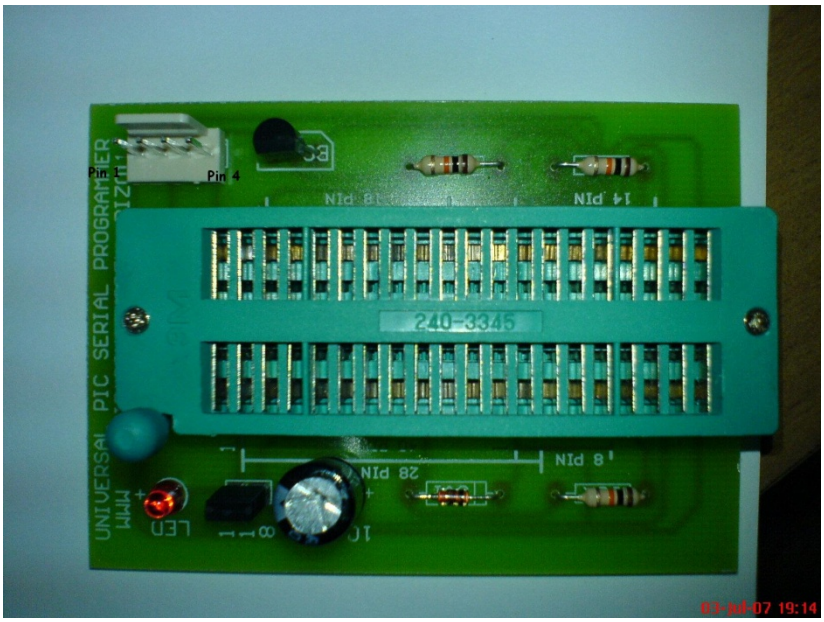


Figure 6.2: The self-made PIC Programmer.

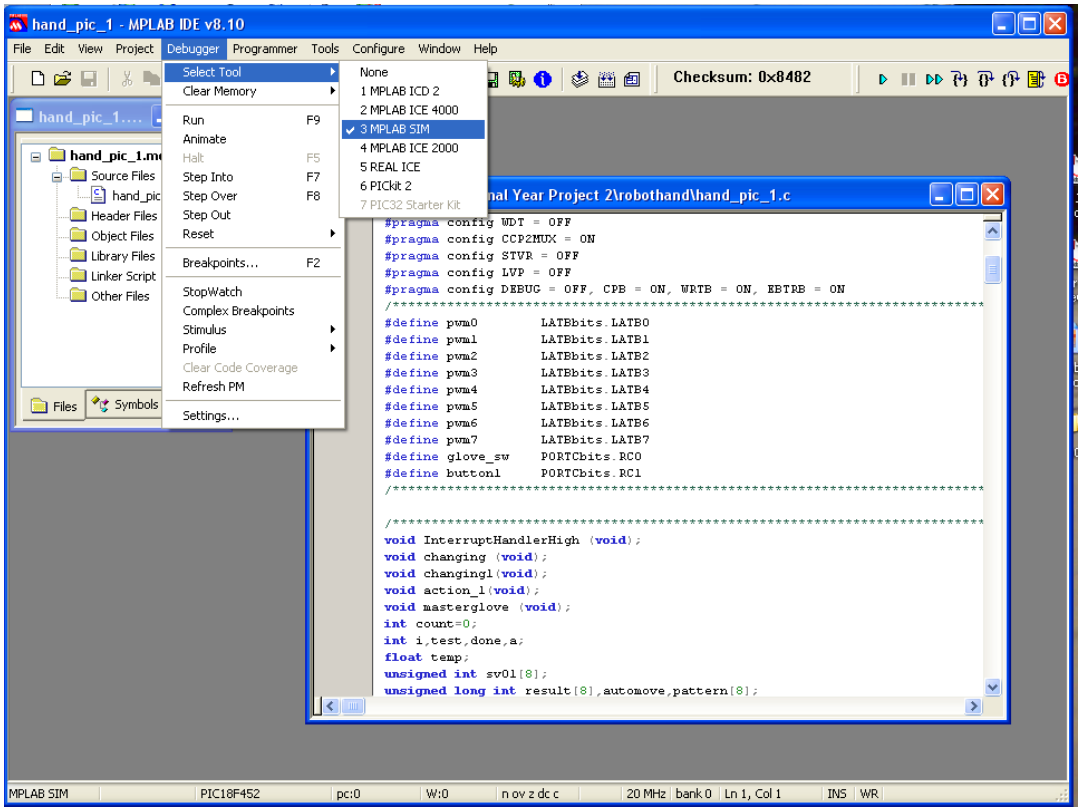


Figure 6.3: The MPLAB IDE programming, Simulating and debugging platform.

During the program writing, testing and debugging process, one of the robot hand's fingers was tested to make sure the finger can move as human finger. At the beginning state, 3 servomotors were debugged so that the right position of the robot fingers can be found. The figure 6.4 shows the process of one finger debugging its position locomotion.

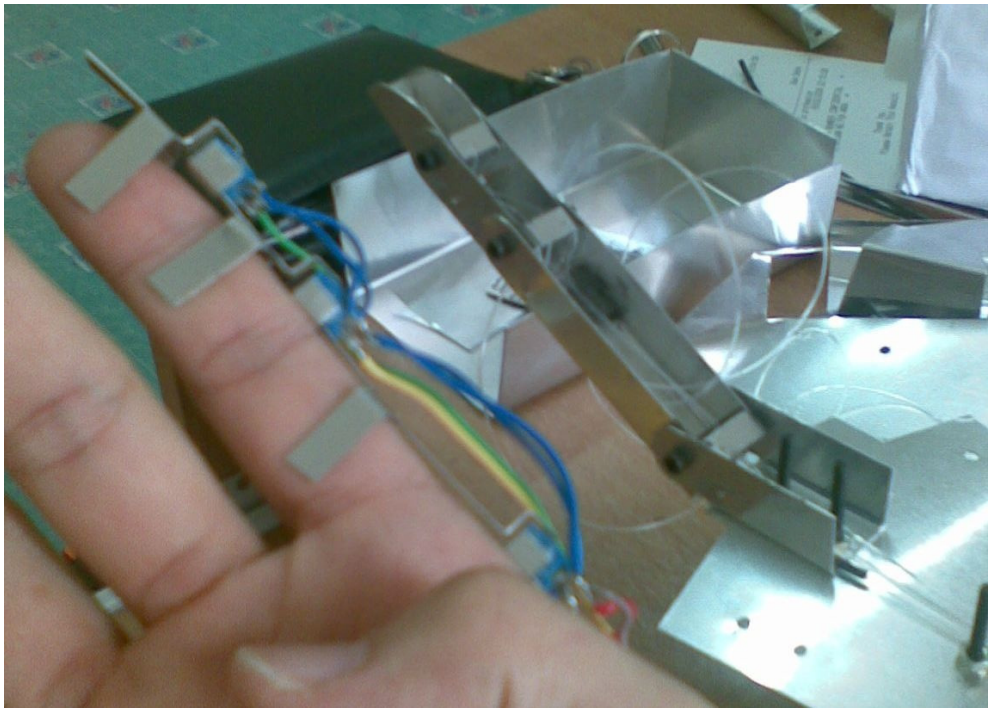


Figure 6.4: Process of one finger debugging its position locomotion

After one finger was successfully debugged, 5 fingers robot hand is built and the same method was used to find the right position locomotion.

6.3 Master glove Controller

The master glove controller is used to control the movement of the robot hand. There are 15 potentiometers built-in into the master glove to generate the different output voltage to the PIC microcontroller. Then the output is used to measure the position of the human hand's fingers. The process of converting analog input to the PWM output was carried out in the PIC microcontroller. This process make the robot hand emulate human hand movement in the real time.

The robot hand can perform a few tasks by controlling it using master glove. The robot hand is successful in emulating human hand and grasping roll materials like water bottle. The figure 6.5 showed the master glove controller and figure 6.6 showed how the master glove controller controls the robot hand.

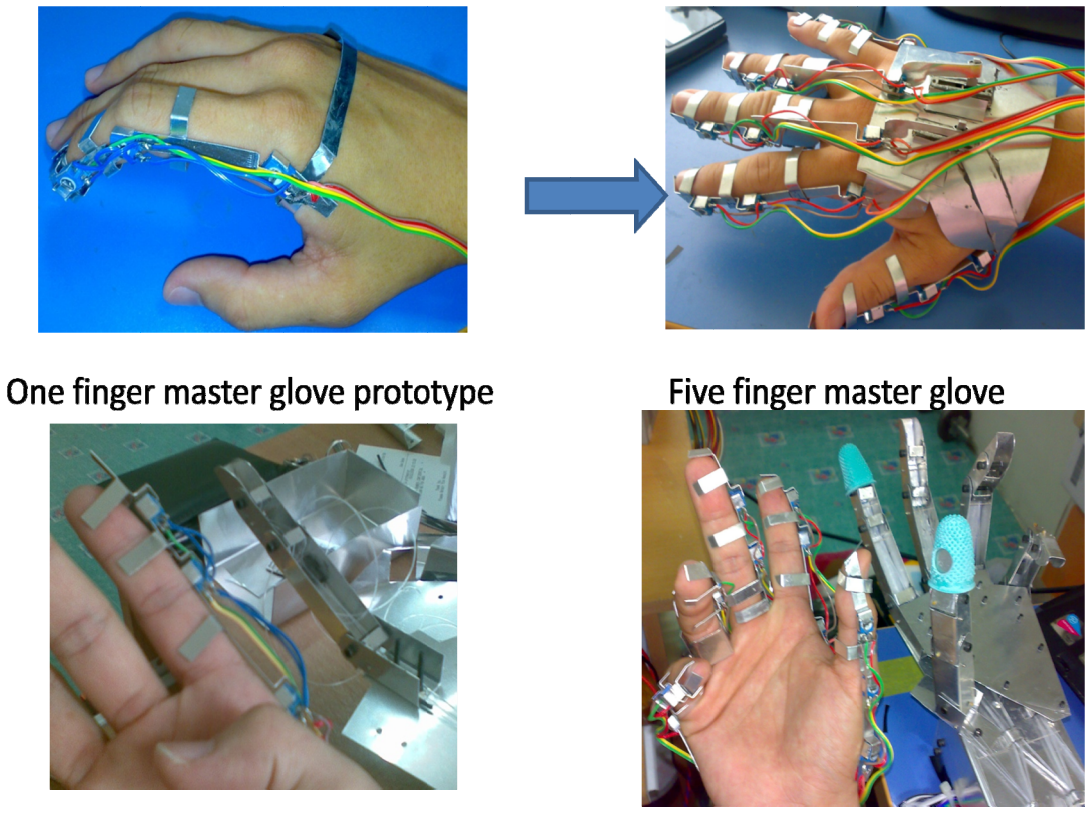
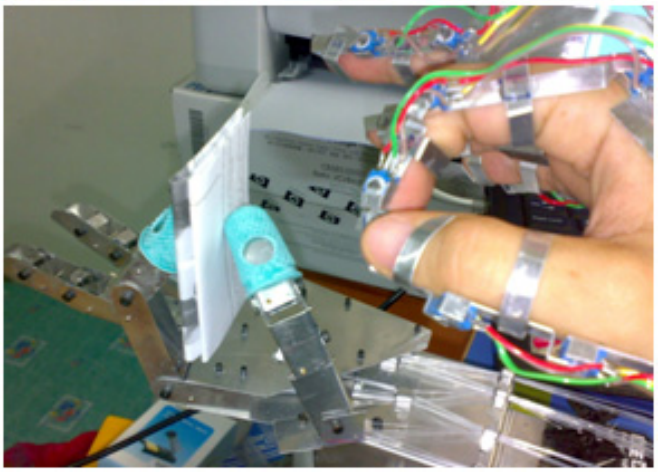


Figure 6.5: Master glove controller

Use master glove to control the robot hand to grasp a bottle.

The robot hand can emulate human hand.



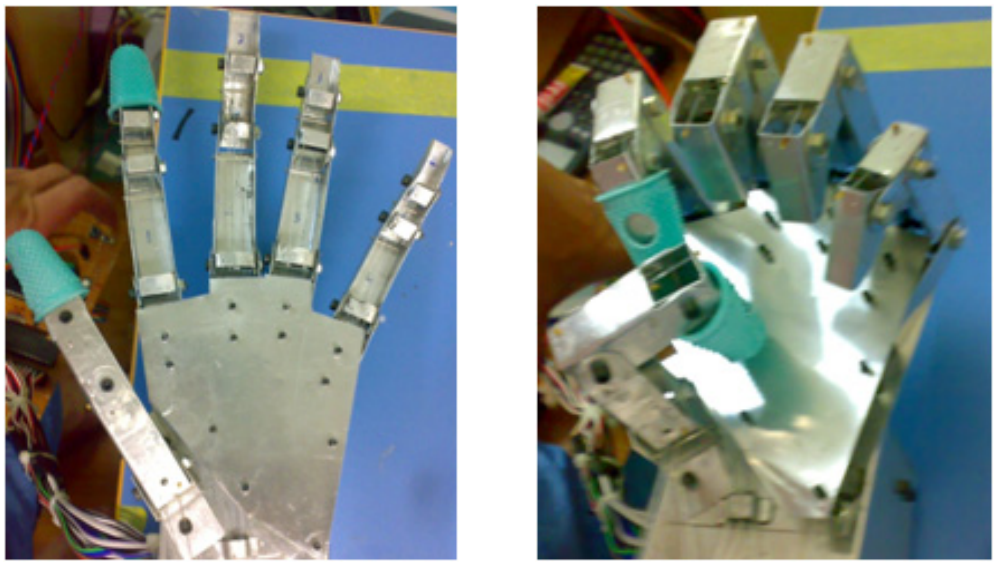
Try to pick a piece of paper.

Figure 6.6: Robot Hand Is Manually Controlled by The Master Glove.

6.4 Autonomous Movement

After switching on the power of the electrical control circuit, the PIC microcontroller will start to run the program. The default state of the robot hand will be which all the fingers are in the straight position. If the button1 in the electrical control circuit has been pushed, the autonomous mode would be activated. In this mode, the robot hand will move following a sequence of instructions. The robot hand will open and close the hand 3 times and fingers bending start from thumb follow by index finger, middle finger, ring finger and the last is pinky finger. After those fingers all close, the hand starts to demonstrate number from one to ten. The figure 6.7 showed the autonomous movement of the robot hand.

1 2



Open and close hand three times

3

Thumb bended

4

Index finger bended

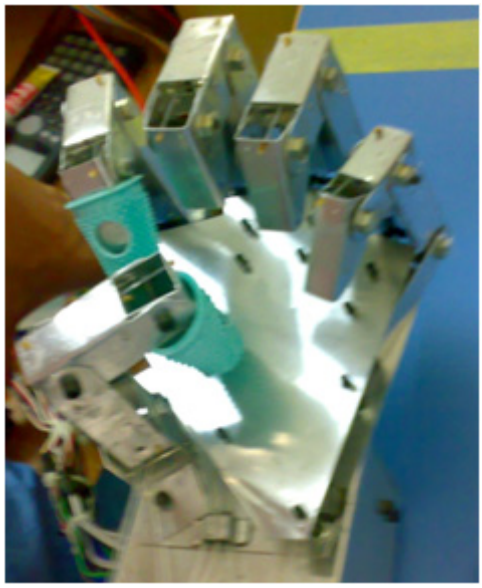
5

Middle finger bended

6

Ring Finger bended

7

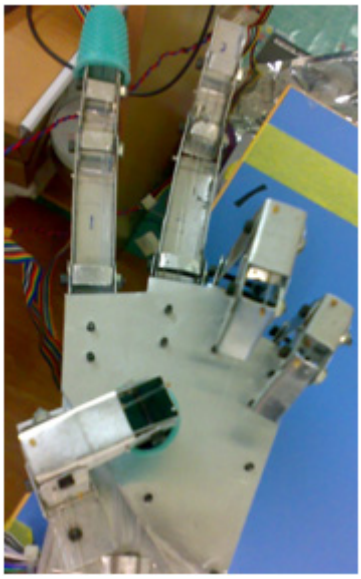


Pinky bended

8

Showing One

9



Showing Two

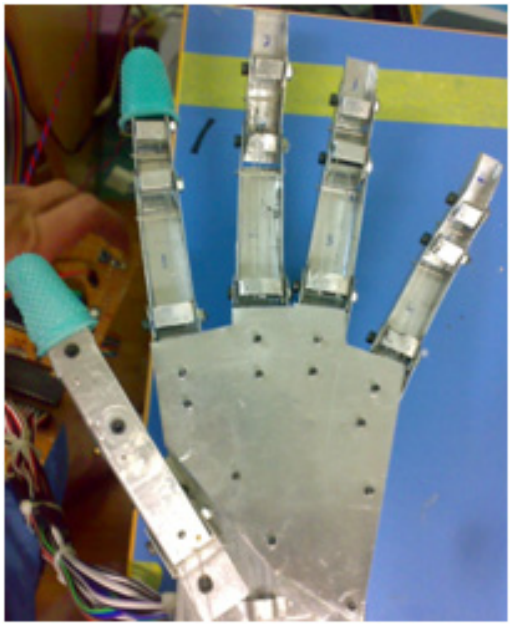
10

Showing Three

11

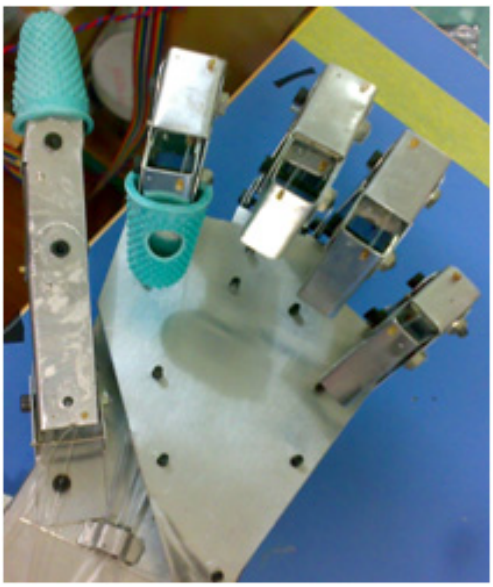
Showing Four

12



Showing Five

13

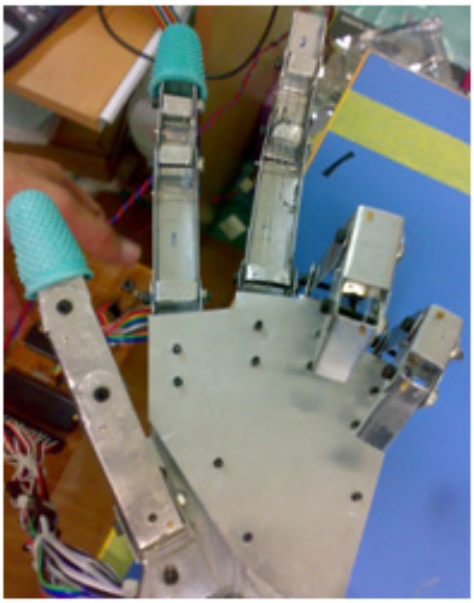


Showing Six

14

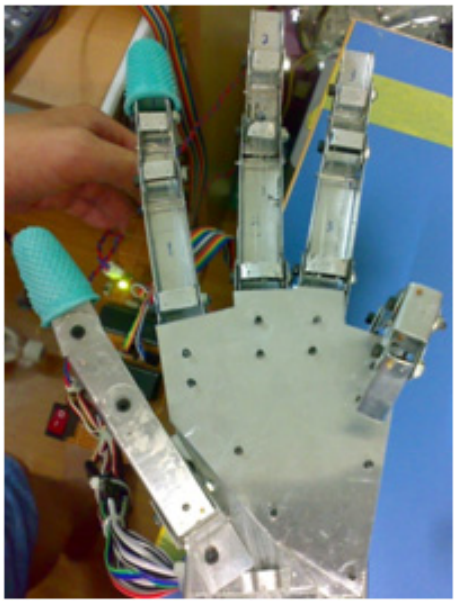
Showing Seven

15



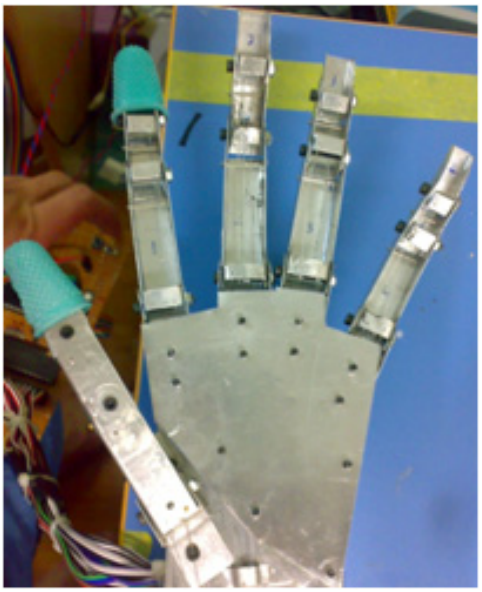
Showing Eight

16



Showing Nine

17



Showing Ten

Figure 6.7: Robot Hand autonomous mode

CHAPTER VII

CONCLUSION AND RECOMMENDATION

7.1 Conclusion

The objectives of this project have been achieved. The robot hand was able to grasp roll objects and emulated human hand. Besides, it is also able to be real time controlled by the master glove. To conclude, the robot hand platform brings a great significance to the humanoid robotic and the invalid. The humanoid robot hand concept can be widely applied into any application and field of research. The humanoid robot is widely research in our real world such as NASA Robonaut Program, Honda Asimo Humanoid Robot, and military research. Hence, the robot hand project is a right-on-time project with a wide array of opportunity scopes in the field of economy, scientific research and design. The knowledge and skills obtained through this project will bring in a lot of benefits and opportunities.

6.1 Recommendation

There are still a lot of space for improvement and enhancement for this 5 fingers robot hand project. Humanoid robot covers a very large field which requires creativity, talent and dynamic mentality to fully optimize the technology, knowledge and inspiration of the nature. The master glove design is a mechanical design while all the sensor used was potentiometer. The mechanical design made the human hand's fingers hard to move, hard to wear and only in 3 degree of freedom for each finger. This can be improved by using Flexible Bend Sensor to make the master glove become suppleness, easy to wear and easy to use.

The servo motor used in this project was in normal size and 16 servo motors were assembled in the forearm. The servo motors fixed the size of human forearm, but the torque was not enough to let the finger to support heavy load. The more powerful air muscle should be applied with a metallic string to enhance the robot hand's fingers movement and its locomotion. Furthermore, the robot hand's finger was 3 degree of freedom, but our human hand has 4 degree of freedom. One degree of freedom needs to be added to improve the robot hand's agility and to make the robot hand can totally emulate human hand.

REFERENCES

[1] Society of Robot (2005-2009). Actuator - Servos [Online]. Available: http://www.societyofrobots.com/actuators_servos.shtml

[2] Society of Robot (2005-2009). BEGINNERS: How to Build Your First Robot Tutorial [Online]. Available: http://www.societyofrobots.com//robot_tutorial.shtml

[3] The University of Texas at Austin. Joint Types [Online]. Available: http://www.robotics.utexas.edu/rrg/learn_more/low_ed/joints/#types

[7] L.Biagiotti,F.Lotti,C.Melchiorri.G.Vassura."Mechatronic Design of Innovative Fingers for Anthropomorphic Robot Hands". IEEE Journal (2003).

[8] Kiyoshi Hoshino,Ichiro Hawabuchi ."Pinching at Finger Tips by Humanoid Robot Hand". IEEE Journal (2005).

[9] L. Biagiotti, E Lotti, C. Melchiom, G. Vassura." Mechatronic Design of Innovative Fingers for Anthropomorphic Robot Hands". IEEE Journal (2003).

[9] Cytron Technologies Sdn. Bhd, "SERVO CONTROLLER User's Manual" (2008)[Online]. Available: http://www.cytron.com.my/listProductCategory.asp?cid=286

[10] PICkit™ 1 Flash Starter Kit User's Guide

[11] Wikipedia, Robotics [Online]. Available: http://en.wikipedia.org/wiki/Robotics

[12] Wikipedia, Humanoid robot [Online]. Available: http://en.wikipedia.org/wiki/Humanoid_robot

[13] Wikipedia, Potentiometer [Online]. Available : http://en.wikipedia.org/wiki/Potentiometer

[14] Chris(2007). Tutorial: Servo Motor Control [Online]. Available: http://www.pyroelectro.com/tutorials/servo_motor/index.html

[15] K.S.Fu, R.C.Gonzalez, C.S.G.Lee. *Robotics control, Sensing, Vision, and Interlligence.* McGRAW-HILL Book Company, 1987

[16] MPLAB® C18 C COMPILER LIBRARIES

APPENDIX A

GANNT CHART

Item	Particulars / Activities	2008															
		Week															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	Registration and department allocation																
2	Decide title																
3	Proposal preparation																
4	Literature review research																
5	Hardware mechanism research																
6	Study PIC																
7	Circuit research																
8	Software and Programming Learning																
9	Prepare for presentation																
10	Presentation of PSM 1																
11	Final Documentation																

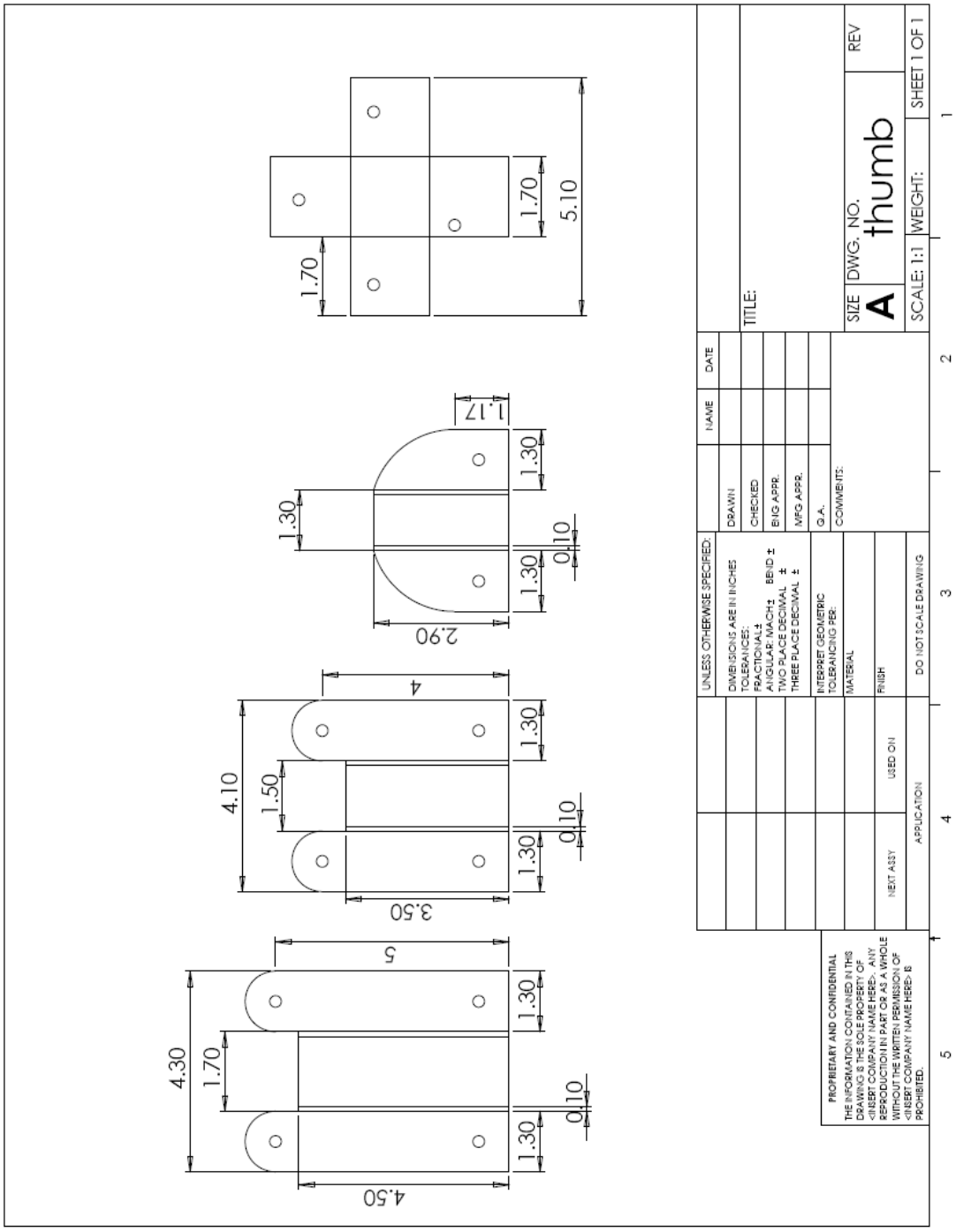
Gantt chart for PSM 1

Item	Particulars / Activities	2009															
		Week															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	Hardware assembly																
2	Main Controller circuit																
3	Combination of Main controller and Servo Controller																
4	Programming																
5	Troubleshooting /Fine Tuning																
6	Preparation for demo/presentation																
7	PSM2 Presentation																
8	Thesis Compilation																

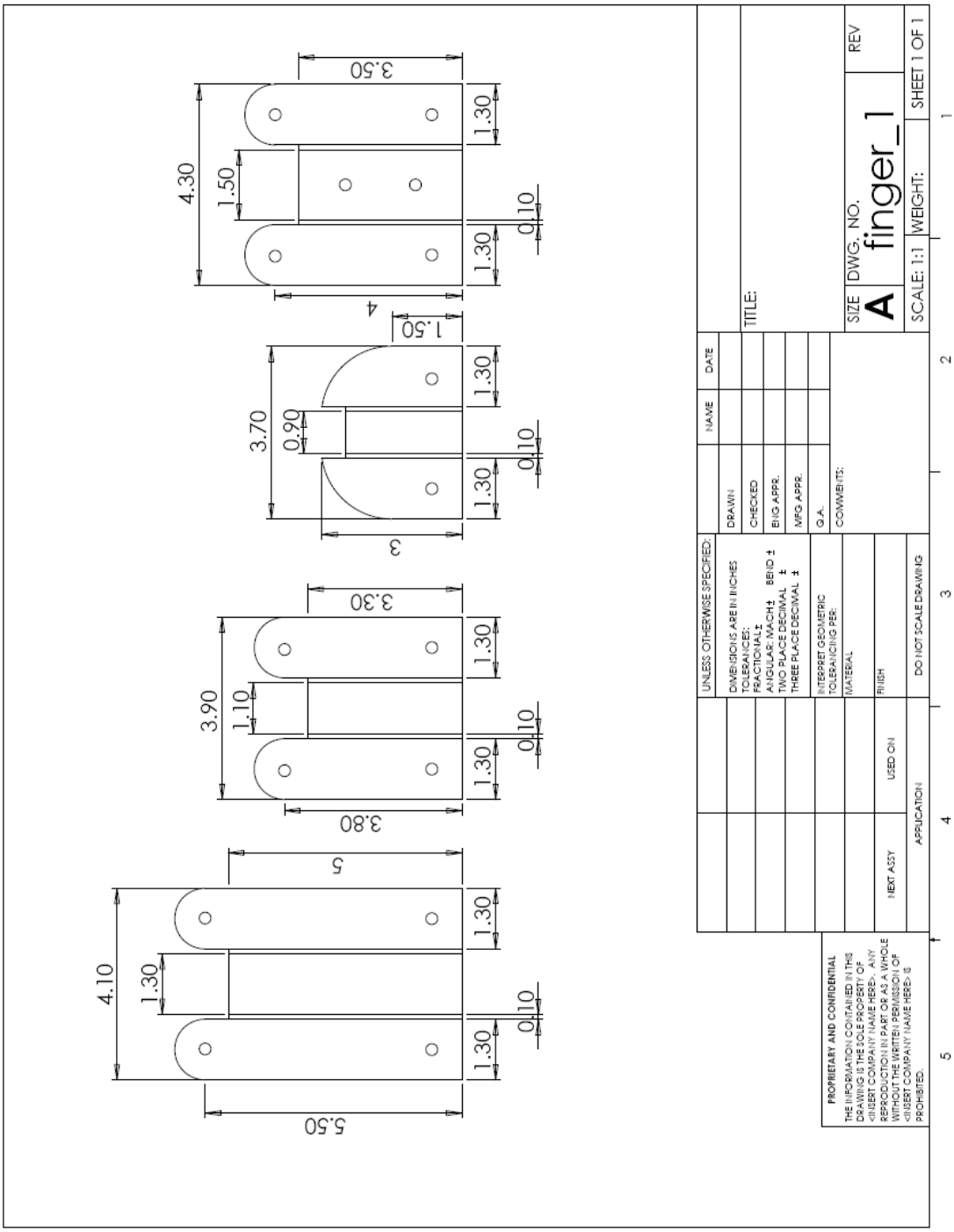
Gantt chart for PSM 2

APPENDIX B

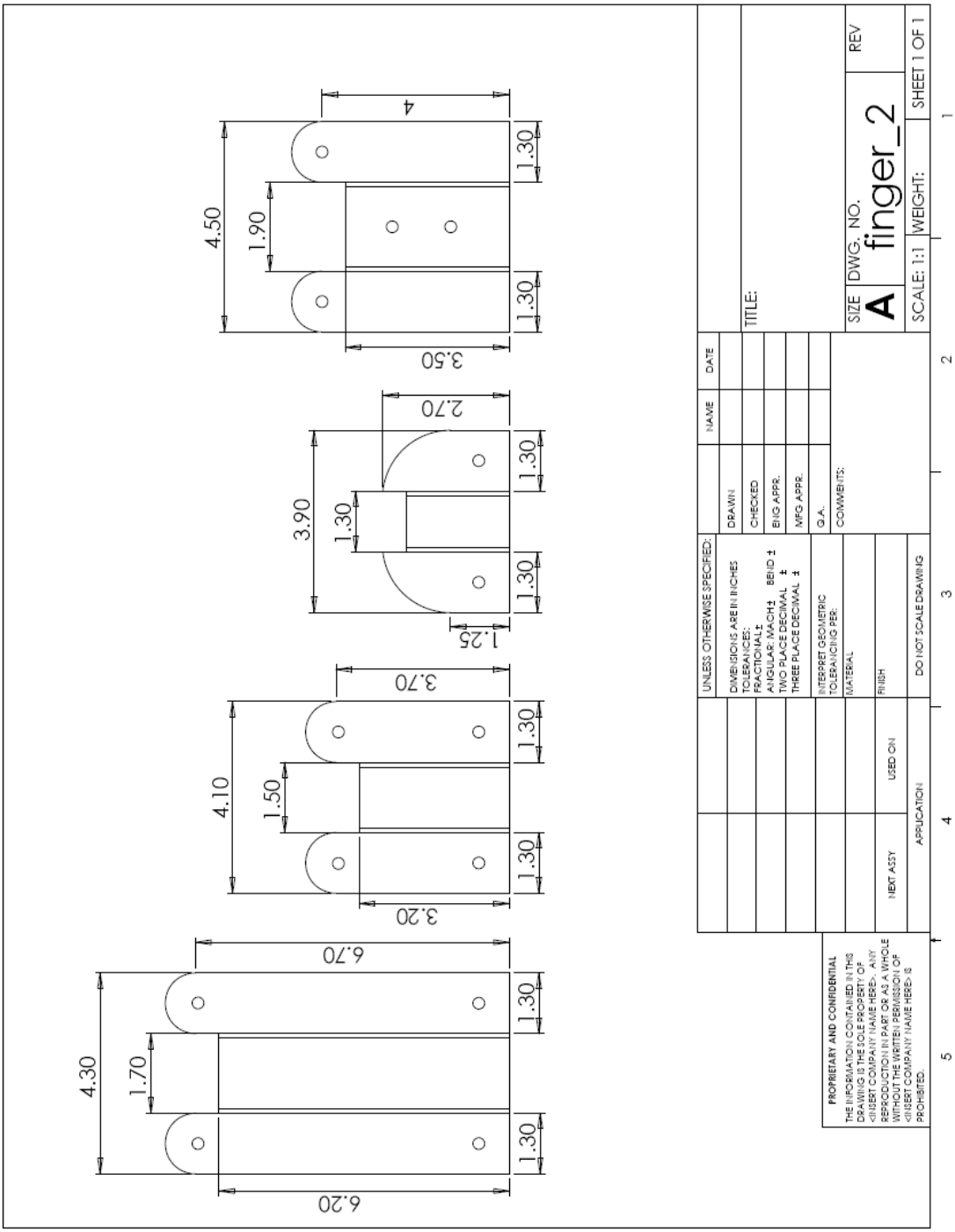
MECHANICAL STRUCTURE LAYOUT



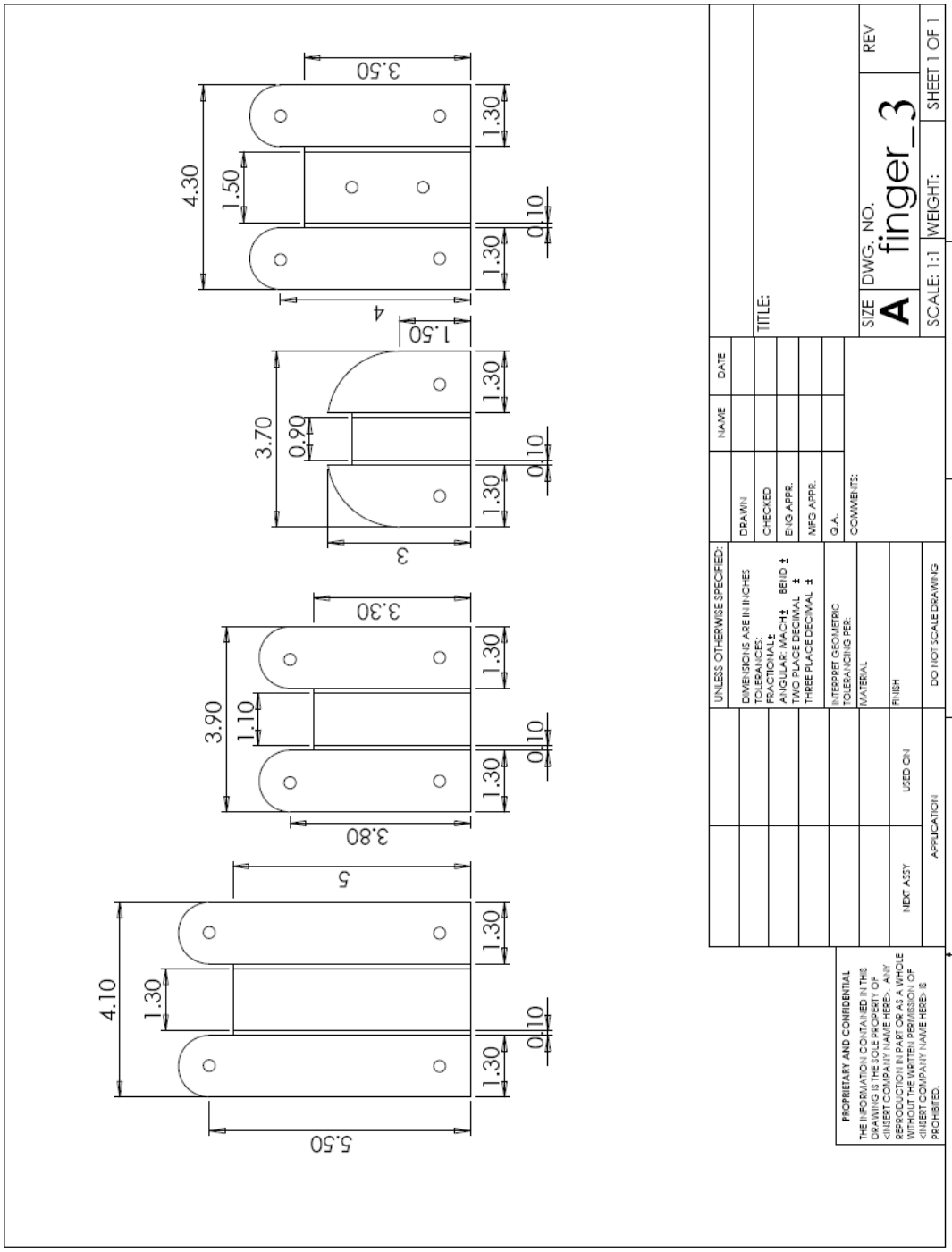
Mechanical Structure Layout for Thumb



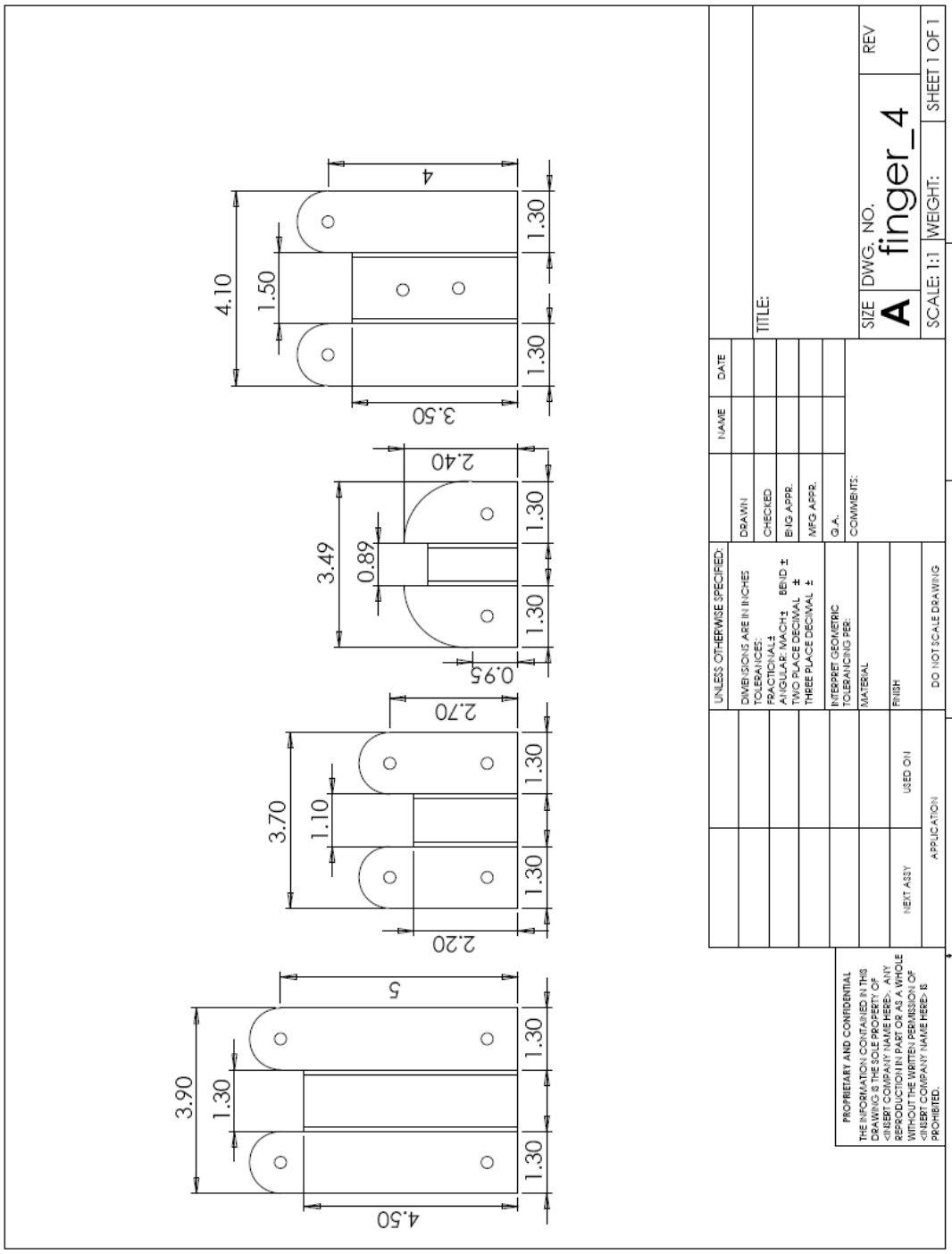
Mechanical Structure Layout for Index Finger



Mechanical Structure Layout for Middle Finger



Mechanical Structure Layout for Ring Finger



Mechanical Structure Layout for Pinky Finger

APPENDIX C

THE SOURCE CODE FOR 5 FINGERS ROBOT HAND WITH PIC 18F452 MICROCONTROLLER

Source Code for Microcontroller 1

```
//Development 5 Fingers Robot Hand Using PIC Microcontroller
//Program for Robot hand's PIC1
//Written by: NG DER LI
//IC No     : 851230-01-5503
//Course    : SEC
//Supervisor: DR. IZZELDIN IMBRAHIM MOHAMED
//


/*****************************************************************/

#include <p18f452.h>
#include <adc.h>
#include <stdlib.h>
#include <timers.h>
#include <delays.h>
/*****************************************************************/

//Configuration

#pragma config OSC = HS
#pragma config OSCS = OFF
#pragma config PWRT = ON
#pragma config BOR = OFF
#pragma config WDT = OFF
#pragma config CCP2MUX = ON
#pragma config STVR = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF, CPB = ON, WRTB = ON, EBTRB = ON
/*****************************************************************/


//Input Output Pin Configuration

#define pwm0        LATBbits.LATB0
#define pwm1        LATBbits.LATB1
#define pwm2        LATBbits.LATB2
#define pwm3        LATBbits.LATB3
#define pwm4        LATBbits.LATB4
#define pwm5        LATBbits.LATB5
#define pwm6        LATBbits.LATB6
#define pwm7        LATBbits.LATB7
#define glove_sw    PORTCbits.RC0
#define button1     PORTCbits.RC1
/*****************************************************************/


/*****************************************************************/

//Function and Parameter Declaration

void InterruptHandlerHigh (void);
void changing (void);
```

```
void changing1(void);
void action_1(void);
void masterglove (void);
int  count=0;
int  i,test,done,a;
float temp;
unsigned int sv01[8];
unsigned long int result[8],automove,pattern[8];

/*****************************************************************/

//INTERRUPT CONTROL
#pragma code InterruptVectorHigh = 0x08     //interrupt pointer
address (0x18 low priority)
void InterruptVectorHigh (void)
{
        _asm        //assembly code starts
        goto InterruptHandlerHigh           //interrupt control
        _endasm //assembly code ends
}
#pragma code
#pragma interrupt InterruptHandlerHigh      //enf.
/*****************************************************************/

void InterruptHandlerHigh()                 // declaration of
InterruptHandler

{//this gets ran when ever the timers flop over from FFFF->0000
        if(INTCONbits.TMR0IF)               //check if TMR0
interrupt flag is set
        {
                WriteTimer0( 0x3CAF );
                WriteTimer1( 0xFC77 );
                count = 0;
                INTCONbits.TMR0IF = 0;      //clear TMR0 flag

        }

        if(PIR1bits.TMR1IF == 1 && PIE1bits.TMR1IE == 1)
//if set controls the first servo
        {
        count++;
                switch(count){
            case 1:     pwm0 = 1;           // First Stage
                        WriteTimer1( sv01[0] );
                        break;
            case 2:     pwm0 = 0; // Servo 1
                        pwm1=1;
                        WriteTimer1( sv01[1] );
                        break;
            case 3:     pwm1 = 0;
                        pwm2=1;
                        WriteTimer1( sv01[2] );
```

```
                    break;
            case 4:     pwm2=0;
                        pwm3=1;
                        WriteTimer1( sv01[3] );
                        break;
            case 5:     pwm3=0;
                        pwm4=1;
                        WriteTimer1( sv01[4] );
                        break;
            case 6:     pwm4=0;
                        pwm5=1;
                        WriteTimer1( sv01[5] );
                        break;
            case 7:     pwm5=0;
                        pwm6=1;
                        WriteTimer1( sv01[6] );
                        break;
            case 8:     pwm6=0;
                        pwm7=1;
                        WriteTimer1( sv01[7] );
                        break;
            case 9:     pwm7=0;
                        WriteTimer1( 0 );
                        break;
                }

                PIR1bits.TMR1IF = 0;//clear Timer1 flag
                PIE1bits.TMR1IE = 1;//clear Timer1 enable flag set
to zero
        }

    INTCONbits.GIE = 1;             //re-enable all interrupts
}

/*****************************************************************/

//Main Program

void main(void)
{

TRISA = 0xFF;                       // PORTA is input
TRISB =  0b00000000;                //Port RC2 is the output
TRISC = 0B11111111;
RCON = 0b000000000;
INTCON = 0b10100000;
OpenTimer0( TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_2 );
OpenTimer1( TIMER_INT_ON & T1_16BIT_RW & T1_SOURCE_INT & T1_PS_1_1
& T1_OSC1EN_OFF & T1_SYNC_EXT_OFF );
WriteTimer0( 0x3CAF );              //Trigger Interrupted after
20mS
WriteTimer1( 62535 );
                                    //This is just a small initial
```

```
delay chosen at random.

while (1)
{
if(glove_sw==0)
{                                       //Master glove control mode
masterglove();
changing1();
}
else if (button1==0)
{                                       //Autonomous mode
automove=520;
test=0;
done=1;
action_1();
}
else
{                                       //Default value for pwm
result[0]=result[1]=result[2]=result[3]=result[4]=result[5]=result[6
]=result[7]=530;
changing();
}

}
}

/*********************************************************************/

//Changing the voltage value to timer interrupt value

void changing(void)
{
for(i=0;i<8;i++)
    {
    if(80<result[i]<530)
        {
        temp=(result[i]*4500/425)+56955;                // Formula
        sv01[i]=temp;
        if (sv01[i]<58035)
            sv01[i]=58035;
        else if (sv01[i]>62535)
            sv01[i]=62535;
        }
    else
        {
        sv01[i]=62535;
        }
    }

}
void changing1(void)
{
for(i=0;i<8;i++)
```

```
    {
    if(80<result[i]<530)
        {
        if(i==5)
        {
            temp=(result[5]*4500/164)+49062;            // Formula
            sv01[5]=temp;
            if (sv01[5]<58035)
                sv01[5]=58035;
            else if (sv01[5]>62535)
                sv01[5]=62535;
        }
        else
        {
            temp=(result[i]*4500/425)+56955;            // Formula
            sv01[i]=temp;
            if (sv01[i]<58035)
                sv01[i]=58035;
            else if (sv01[i]>62535)
                sv01[i]=62535;
        }
        }
    else
        {
        sv01[i]=62535;
        }
    }

}
/*********************************************************************/

//Autonomous mode

void action_1(void)
{
if(done==1)
{
a=0;
while (a<3)
{

result[0]=result[1]=result[2]=result[3]=result[4]=result[5]=result[6
]=result[7]=automove;
    changing();
    Delay10KTCYx(1);
    if (automove>80&&test==0)
        automove=automove-1;
    else if(automove<530)
        {
        test=1;
        automove=automove+1;
        }
    else
```

```
        {
        test=0;
        a++;
        if (a==3)
        done=0;
        }
}
}

if (done==0)
{
test=1;
automove=530;



    while (automove>80)
    {
    result[7]=result[4]=result[5]=automove;
    automove=automove-1;
    changing();
    Delay10KTCYx(1);
    }
automove=530;
    while (automove>80)
    {
    result[2]=result[1]=result[0]=automove;
    automove=automove-1;
    changing();
    Delay10KTCYx(1);
    }
automove=530;
    while (automove>80)
    {
    result[3]=automove;
    automove=automove-1;
    changing();
    Delay10KTCYx(1);
    }
for(i=0;i<4;i++)
{
Delay10KTCYx(255);
}
Delay10KTCYx(40);
while (automove<530)
    {
    result[2]=result[1]=result[0]=automove;
    automove=automove+1;
    changing();
    Delay10KTCYx(1);
    }
automove=80;
while (automove<530)
```

```
    {
    result[3]=automove;
    automove=automove+1;
    changing();
    Delay10KTCYx(1);
    }
for(i=0;i<6;i++)
{
Delay10KTCYx(255);
}
Delay10KTCYx(60);
automove=80;
while (automove<530)
    {
    result[7]=result[4]=result[5]=automove;
    automove=automove+1;
    changing();
    Delay10KTCYx(1);
    }
while (automove>80)
    {
    result[0]=result[1]=result[2]=result[3]=result[6]=automove;
    automove=automove-1;
    changing();
    Delay10KTCYx(1);
    }
while (automove<530)
    {
    result[2]=result[1]=result[0]=automove;
    automove=automove+1;
    changing();
    Delay10KTCYx(1);
    }
automove=80;
while (automove<530)
    {
    result[3]=automove;
    automove=automove+1;
    changing();
    Delay10KTCYx(1);
    }
}
}
/*****************************************************************/

//Read and process Value from Analog input

void masterglove (void)
{
OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH0 & ADC_INT_OFF );
Delay10TCYx( 5);                        // Delay for 50TCY
ConvertADC();                           // Start conversion
```

```
while( BusyADC() );                         // Wait for completion
result[0] = ReadADC();                      // Read result
CloseADC();                                 // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH1 & ADC_INT_OFF );
Delay10TCYx( 5);                            // Delay for 50TCY
ConvertADC();                               // Start conversion
while( BusyADC() );                         // Wait for completion
result[1] = ReadADC();                      // Read result
CloseADC();                                 // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH2 & ADC_INT_OFF );
Delay10TCYx( 5);                            // Delay for 50TCY
ConvertADC();                               // Start conversion
while( BusyADC() );                         // Wait for completion
result[2] = ReadADC();                      // Read result
CloseADC();                                 // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH3 & ADC_INT_OFF );
Delay10TCYx( 5);                            // Delay for 50TCY
ConvertADC();                               // Start conversion
while( BusyADC() );                         // Wait for completion
result[3] = ReadADC();                      // Read result
CloseADC();                                 // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH4 & ADC_INT_OFF );
Delay10TCYx( 5);                            // Delay for 50TCY
ConvertADC();                               // Start conversion
while( BusyADC() );                         // Wait for completion
result[4] = ReadADC();                      // Read result
CloseADC();                                 // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH5 & ADC_INT_OFF );
Delay10TCYx( 5);                            // Delay for 50TCY
ConvertADC();                               // Start conversion
while( BusyADC() );                         // Wait for completion
result[5] = ReadADC();                      // Read result
CloseADC();                                 // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH6 & ADC_INT_OFF );
Delay10TCYx( 5);                            // Delay for 50TCY
```

```
ConvertADC();                                   // Start conversion
while( BusyADC() );                             // Wait for completion
result[6] = ReadADC();                          // Read result
CloseADC();                                     // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH7 & ADC_INT_OFF );
Delay10TCYx( 5);                                // Delay for 50TCY
ConvertADC();                                   // Start conversion
while( BusyADC() );                             // Wait for completion
result[7] = ReadADC();                          // Read result
CloseADC();                                     // Disable A/D
converter
}
```

Source Code for Microcontroller 2

```
//Development 5 Fingers Robot Hand Using PIC Microcontroller
//Program for Robot hand's PIC2
//Written by: NG DER LI
//IC No     : 851230-01-5503
//Course    : SEC
//Supervisor: DR. IZZELDIN IMBRAHIM MOHAMED
//
/*****************************************************************/

#include <p18f452.h>
#include <adc.h>
#include <stdlib.h>
#include <timers.h>
#include <delays.h>
/*****************************************************************/

//Configuration
#pragma config OSC = HS
#pragma config OSCS = OFF
#pragma config PWRT = ON
#pragma config BOR = OFF
#pragma config WDT = OFF
#pragma config CCP2MUX = ON
#pragma config STVR = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF, CPB = ON, WRTB = ON, EBTRB = ON
/*****************************************************************/

//Input Output Pin Configuration
#define pwm0        LATBbits.LATB0
#define pwm1        LATBbits.LATB1
#define pwm2        LATBbits.LATB2
#define pwm3        LATBbits.LATB3
#define pwm4        LATBbits.LATB4
#define pwm5        LATBbits.LATB5
#define pwm6        LATBbits.LATB6
#define pwm7        LATBbits.LATB7
#define glove_sw    PORTCbits.RC0
#define button1     PORTCbits.RC1
/*****************************************************************/


/*****************************************************************/

//Function and Parameter Declaration
void InterruptHandlerHigh (void);
void changing (void);
void action_1(void);
void masterglove (void);
int count=0;
int i,test,done,a;
float temp;
unsigned int sv01[8];
```

```
unsigned long int result[8],automove,pattern[8];
/*****************************************************************/

//INTERRUPT CONTROL
#pragma code InterruptVectorHigh = 0x08     //interrupt pointer
address (0x18 low priority)
void InterruptVectorHigh (void)
{
        _asm        //assembly code starts
        goto InterruptHandlerHigh          //interrupt control
        _endasm //assembly code ends
}
#pragma code
#pragma interrupt InterruptHandlerHigh      //enf.
/*****************************************************************/

void InterruptHandlerHigh()       // declaration of InterruptHandler
{//this gets ran when ever the timers flop over from FFFF->0000
        if(INTCONbits.TMR0IF) //check if TMR0 interrupt flag is set
        {
                WriteTimer0( 0x3CAF );
                WriteTimer1( 0xFC77 );
                count = 0;
                INTCONbits.TMR0IF = 0;      //clear TMR0 flag

        }
        if(PIR1bits.TMR1IF == 1 && PIE1bits.TMR1IE == 1)
        //if set controls the first servo
        {
        count++;
                switch(count){
            case 1:     pwm0 = 1;           // First Stage
                        WriteTimer1( sv01[0] );
                        break;
            case 2:     pwm0 = 0;           // Servo 1
                        pwm1=1;
                        WriteTimer1( sv01[1] );
                        break;
            case 3:     pwm1 = 0;
                        pwm2=1;
                        WriteTimer1( sv01[2] );
                        break;
            case 4:     pwm2=0;
                        pwm3=1;
                        WriteTimer1( sv01[3] );
                        break;
            case 5:     pwm3=0;
                        pwm4=1;
                        WriteTimer1( sv01[4] );
                        break;
            case 6:     pwm4=0;
                        pwm5=1;
                        WriteTimer1( sv01[5] );
```

```
                    break;
            case 7:     pwm5=0;
                        pwm6=1;
                        WriteTimer1( sv01[6] );
                        break;
            case 8:     pwm6=0;
                        pwm7=1;
                        WriteTimer1( sv01[7] );
                        break;
            case 9:     pwm7=0;
                        WriteTimer1( 0 );
                        break;
                }

                PIR1bits.TMR1IF = 0;//clear Timer1 flag
                PIE1bits.TMR1IE = 1;//clear Timer1 enable flag set
to zero
        }

    INTCONbits.GIE = 1;             //re-enable all interrupts
}
/*****************************************************************/

void main(void)
{

TRISA = 0xFF;                       // PORTA is input
TRISB =  0b00000000;                //Port RC2 is the output
TRISC = 0B11111111;
RCON = 0b000000000;
INTCON = 0b10100000;
OpenTimer0( TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_2 );
OpenTimer1( TIMER_INT_ON & T1_16BIT_RW & T1_SOURCE_INT & T1_PS_1_1
& T1_OSC1EN_OFF & T1_SYNC_EXT_OFF );
WriteTimer0( 0x3CAF );  //Trigger Interrupted after 20mS
WriteTimer1( 62535 );   //This is just a small initial delay
chosen at random.

while (1)
{
if(glove_sw==0)
{
masterglove();
changing();
}
else if (button1==0)
{
automove=520;
test=0;
done=1;
action_1();
}
else
```

```
{
result[0]=result[1]=result[2]=result[3]=result[4]=result[5]=result[6
]=result[7]=530;
changing();
}
}
}

/*****************************************************************/

void changing(void)
{
for(i=0;i<8;i++)
    {
    if(80<result[i]<530)
        {
        temp=(result[i]*4500/425)+56955;     // Formula
        sv01[i]=temp;
        if (sv01[i]<58035)
            sv01[i]=58035;
        else if (sv01[i]>62535)
            sv01[i]=62535;
        }
    else
        {
        sv01[i]=62535;
        }
    }

}
/*****************************************************************/

void action_1(void)
{
if(done==1)
{
a=0;
while (a<3)
{

result[0]=result[1]=result[2]=result[3]=result[4]=result[5]=result[6
]=result[7]=automove;
    changing();
    Delay10KTCYx(1);
    if (automove>80&&test==0)
        automove=automove-1;
    else if(automove<530)
        {
        test=1;
        automove=automove+1;
        }
    else
        {
```

```
            test=0;
            a++;
            if (a==3)
            done=0;
            }
}
}

if (done==0)
{
for(i=0;i<6;i++)
{
Delay10KTCYx(255);
}
Delay10KTCYx(60);
test=1;
automove=530;
     while (automove>80)
     {
     result[7]=result[6]=automove;
     automove=automove-1;
     changing();
     Delay10KTCYx(1);
     }
automove=530;
     while (automove>80)
     {
     result[3]=result[4]=result[5]=automove;
     automove=automove-1;
     changing();
     Delay10KTCYx(1);
     }
automove=530;
     while (automove>80)
     {
     result[0]=result[1]=result[2]=automove;
     automove=automove-1;
     changing();
     Delay10KTCYx(1);
     }
for(i=0;i<2;i++)
{
Delay10KTCYx(255);
}
Delay10KTCYx(20);
while (automove<530)
     {
     result[7]=result[6]=automove;
     automove=automove+1;
     changing();
     Delay10KTCYx(1);
     }
automove=80;
```

```
while (automove<530)
     {
     result[3]=result[4]=result[5]=automove;
     automove=automove+1;
     changing();
     Delay10KTCYx(1);
     }
automove=80;
while (automove<530)
     {
     result[0]=result[1]=result[2]=automove;
     automove=automove+1;
     changing();
     Delay10KTCYx(1);
     }
for(i=0;i<2;i++)
{
Delay10KTCYx(255);
}
Delay10KTCYx(20);

while (automove>80)
     {

result[0]=result[1]=result[2]=result[3]=result[4]=result[5]=result[6
]=result[7]=automove;
     automove=automove-1;
     changing();
     Delay10KTCYx(1);
     }
for(i=0;i<2;i++)
{
Delay10KTCYx(255);
}
Delay10KTCYx(20);
while (automove<530)
     {
     result[7]=result[6]=automove;
     automove=automove+1;
     changing();
     Delay10KTCYx(1);
     }
automove=80;
while (automove<530)
     {
     result[3]=result[4]=result[5]=automove;
     automove=automove+1;
     changing();
     Delay10KTCYx(1);
     }
automove=80;
while (automove<530)
     {
```

```
    result[0]=result[1]=result[2]=automove;
    automove=automove+1;
    changing();
    Delay10KTCYx(1);
    }
}
}
/*****************************************************************/

void masterglove(void)
{
OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH0 & ADC_INT_OFF );
Delay10TCYx( 5);                        // Delay for 50TCY
ConvertADC();                           // Start conversion
while( BusyADC() );                     // Wait for completion
result[0] = ReadADC();                  // Read result
CloseADC();                             // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH1 & ADC_INT_OFF );
Delay10TCYx( 5);                        // Delay for 50TCY
ConvertADC();                           // Start conversion
while( BusyADC() );                     // Wait for completion
result[1] = ReadADC();                  // Read result
CloseADC();                             // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH2 & ADC_INT_OFF );
Delay10TCYx( 5);                        // Delay for 50TCY
ConvertADC();                           // Start conversion
while( BusyADC() );                     // Wait for completion
result[2] = ReadADC();                  // Read result
CloseADC();                             // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH3 & ADC_INT_OFF );
Delay10TCYx( 5);                        // Delay for 50TCY
ConvertADC();                           // Start conversion
while( BusyADC() );                     // Wait for completion
result[3] = ReadADC();                  // Read result
CloseADC();                             // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH4 & ADC_INT_OFF );
Delay10TCYx( 5);                        // Delay for 50TCY
ConvertADC();                           // Start conversion
while( BusyADC() );                     // Wait for completion
result[4] = ReadADC();                  // Read result
```

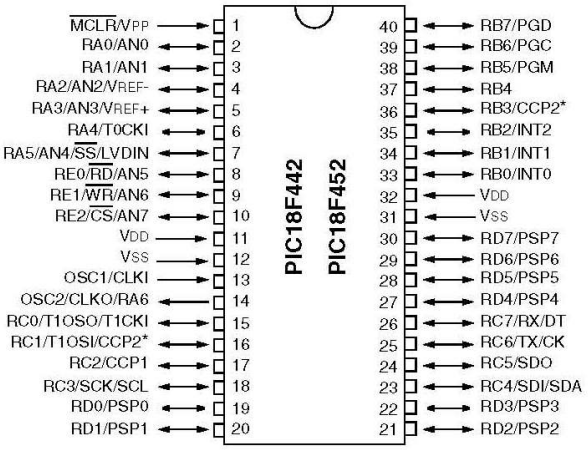
```
CloseADC();                                    // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH5 & ADC_INT_OFF );
Delay10TCYx( 5);                               // Delay for 50TCY
ConvertADC();                                  // Start conversion
while( BusyADC() );                            // Wait for completion
result[5] = ReadADC();                         // Read result
CloseADC();                                    // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH6 & ADC_INT_OFF );
Delay10TCYx( 5);                               // Delay for 50TCY
ConvertADC();                                  // Start conversion
while( BusyADC() );                            // Wait for completion
result[6] = ReadADC();                         // Read result
CloseADC();                                    // Disable A/D
converter

OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH7 & ADC_INT_OFF );
Delay10TCYx( 5);                               // Delay for 50TCY
ConvertADC();                                  // Start conversion
while( BusyADC() );                            // Wait for completion
result[7] = ReadADC();                         // Read result
CloseADC();                                    // Disable A/D
converter

}
```

APPENDIX C

DATASHEET COMPONENT

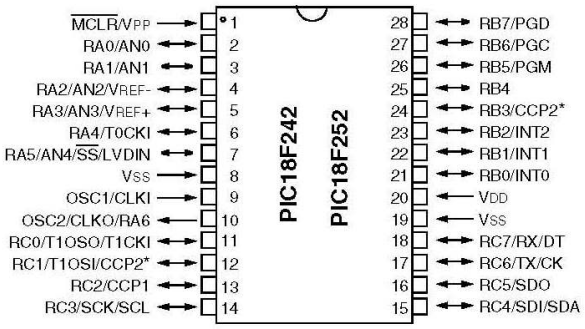
Pin Diagrams (Cont.'d)

DIP



Note: Pin compatible with 40-pin PIC16C7X devices.

DIP, SOIC



* RB3 is the alternate pin for the CCP2 pin multiplexing.

3.0 RESET

The PIC18FXXX differentiates between various kinds of RESET:

a) Power-on Reset (POR)
b) $\overline{\text{MCLR}}$ Reset during normal operation
c) $\overline{\text{MCLR}}$ Reset during SLEEP
d) Watchdog Timer (WDT) Reset (during normal operation)
e) Programmable Brown-out Reset (BOR)
f) RESET Instruction
g) Stack Full Reset
h) Stack Underflow Reset

Most registers are unaffected by a RESET. Their status is unknown on POR and unchanged by all other RESETS. The other registers are forced to a "RESET state" on Power-on Reset, $\overline{\text{MCLR}}$, WDT Reset, Brown-out Reset, $\overline{\text{MCLR}}$ Reset during SLEEP and by the RESET instruction.

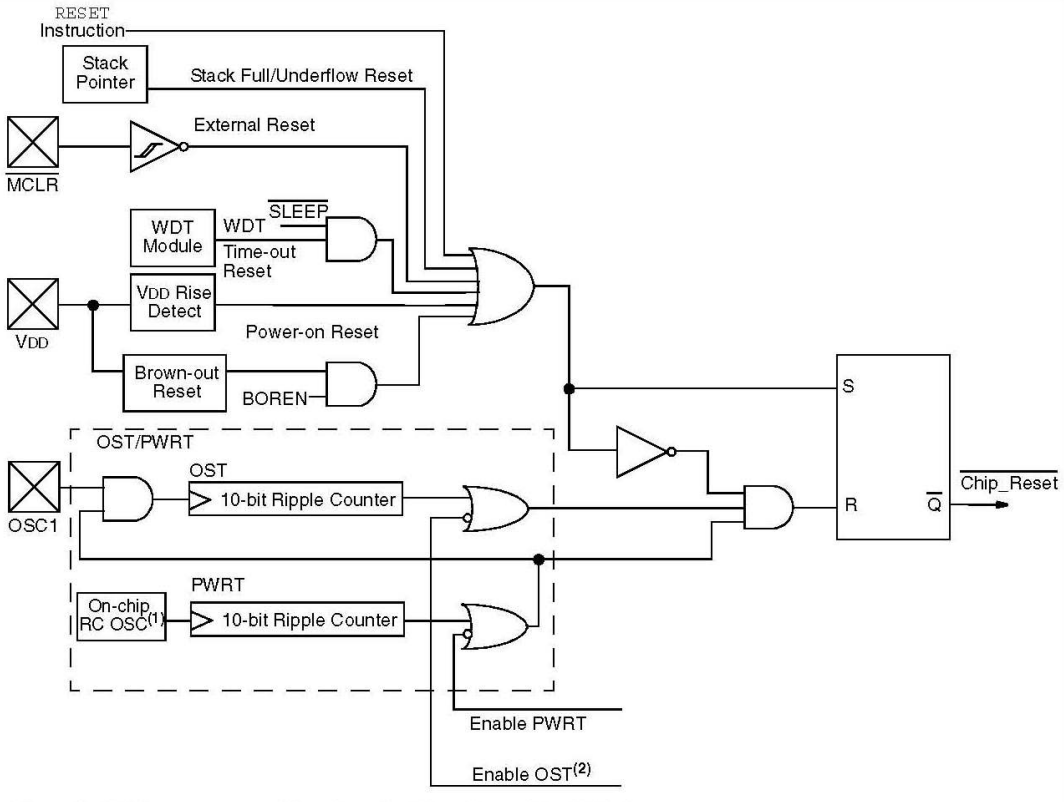
Most registers are not affected by a WDT wake-up, since this is viewed as the resumption of normal operation. Status bits from the RCON register, $\overline{\text{RI}}$, $\overline{\text{TO}}$, $\overline{\text{PD}}$, $\overline{\text{POR}}$ and $\overline{\text{BOR}}$, are set or cleared differently in different RESET situations, as indicated in Table 3-2. These bits are used in software to determine the nature of the RESET. See Table 3-3 for a full description of the RESET states of all registers.

A simplified block diagram of the On-Chip Reset Circuit is shown in Figure 3-1.

The Enhanced MCU devices have a $\overline{\text{MCLR}}$ noise filter in the $\overline{\text{MCLR}}$ Reset path. The filter will detect and ignore small pulses.

The $\overline{\text{MCLR}}$ pin is not driven low by any internal RESETS, including the WDT.

FIGURE 3-1: SIMPLIFIED BLOCK DIAGRAM OF ON-CHIP RESET CIRCUIT



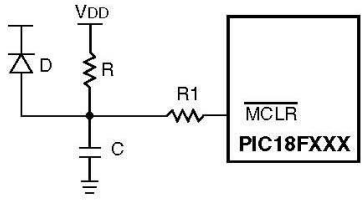
Note 1: This is a separate oscillator from the RC oscillator of the CLKI pin.
2: See Table 3-1 for time-out situations.

3.1 Power-On Reset (POR)

A Power-on Reset pulse is generated on-chip when VDD rise is detected. To take advantage of the POR circuitry, just tie the MCLR pin directly (or through a resistor) to VDD. This will eliminate external RC components usually needed to create a Power-on Reset delay. A minimum rise rate for VDD is specified (parameter D004). For a slow rise time, see Figure 3-2.

When the device starts normal operation (i.e., exits the RESET condition), device operating parameters (voltage, frequency, temperature, etc.) must be met to ensure operation. If these conditions are not met, the device must be held in RESET until the operating conditions are met.

FIGURE 3-2: EXTERNAL POWER-ON RESET CIRCUIT (FOR SLOW VDD POWER-UP)



Note 1: External Power-on Reset circuit is required only if the VDD power-up slope is too slow. The diode D helps discharge the capacitor quickly when VDD powers down.

2: R < 40 kΩ is recommended to make sure that the voltage drop across R does not violate the device's electrical specification.

3: R1 = 100Ω to 1 kΩ will limit any current flowing into MCLR from external capacitor C, in the event of MCLR/VPP pin breakdown due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS).

3.2 Power-up Timer (PWRT)

The Power-up Timer provides a fixed nominal time-out (parameter 33) only on power-up from the POR. The Power-up Timer operates on an internal RC oscillator. The chip is kept in RESET as long as the PWRT is active. The PWRT's time delay allows VDD to rise to an acceptable level. A configuration bit is provided to enable/disable the PWRT.

The power-up time delay will vary from chip-to-chip due to VDD, temperature and process variation. See DC parameter D033 for details.

3.3 Oscillator Start-up Timer (OST)

The Oscillator Start-up Timer (OST) provides a 1024 oscillator cycle (from OSC1 input) delay after the PWRT delay is over (parameter 32). This ensures that the crystal oscillator or resonator has started and stabilized.

The OST time-out is invoked only for XT, LP and HS modes and only on Power-on Reset or wake-up from SLEEP.

3.4 PLL Lock Time-out

With the PLL enabled, the time-out sequence following a Power-on Reset is different from other Oscillator modes. A portion of the Power-up Timer is used to provide a fixed time-out that is sufficient for the PLL to lock to the main oscillator frequency. This PLL lock time-out (TPLL) is typically 2 ms and follows the oscillator start-up time-out (OST).

3.5 Brown-out Reset (BOR)

A configuration bit, BOREN, can disable (if clear/programmed), or enable (if set) the Brown-out Reset circuitry. If VDD falls below parameter D005 for greater than parameter 35, the brown-out situation will reset the chip. A RESET may not occur if VDD falls below parameter D005 for less than parameter 35. The chip will remain in Brown-out Reset until VDD rises above BVDD. If the Power-up Timer is enabled, it will be invoked after VDD rises above BVDD; it then will keep the chip in RESET for an additional time delay (parameter 33). If VDD drops below BVDD while the Power-up Timer is running, the chip will go back into a Brown-out Reset and the Power-up Timer will be initialized. Once VDD rises above BVDD, the Power-up Timer will execute the additional time delay.

3.6 Time-out Sequence

On power-up, the time-out sequence is as follows: First, PWRT time-out is invoked after the POR time delay has expired. Then, OST is activated. The total time-out will vary based on oscillator configuration and the status of the PWRT. For example, in RC mode with the PWRT disabled, there will be no time-out at all. Figure 3-3, Figure 3-4, Figure 3-5, Figure 3-6 and Figure 3-7 depict time-out sequences on power-up.

Since the time-outs occur from the POR pulse, if MCLR is kept low long enough, the time-outs will expire. Bringing MCLR high will begin execution immediately (Figure 3-5). This is useful for testing purposes or to synchronize more than one PIC18FXXX device operating in parallel.

Table 3-2 shows the RESET conditions for some Special Function Registers, while Table 3-3 shows the RESET conditions for all the registers.

© 2006 Microchip Technology Inc.

8.0 INTERRUPTS

The PIC18FXX2 devices have multiple interrupt sources and an interrupt priority feature that allows each interrupt source to be assigned a high priority level or a low priority level. The high priority interrupt vector is at 000008h and the low priority interrupt vector is at 000018h. High priority interrupt events will override any low priority interrupts that may be in progress.

There are ten registers which are used to control interrupt operation. These registers are:

- RCON
- INTCON
- INTCON2
- INTCON3
- PIR1, PIR2
- PIE1, PIE2
- IPR1, IPR2

It is recommended that the Microchip header files supplied with MPLAB® IDE be used for the symbolic bit names in these registers. This allows the assembler/compiler to automatically take care of the placement of these bits within the specified register.

Each interrupt source, except INT0, has three bits to control its operation. The functions of these bits are:

- Flag bit to indicate that an interrupt event occurred
- Enable bit that allows program execution to branch to the interrupt vector address when the flag bit is set
- Priority bit to select high priority or low priority

The interrupt priority feature is enabled by setting the IPEN bit (RCON<7>). When interrupt priority is enabled, there are two bits which enable interrupts globally. Setting the GIEH bit (INTCON<7>) enables all interrupts that have the priority bit set. Setting the GIEL bit (INTCON<6>) enables all interrupts that have the priority bit cleared. When the interrupt flag, enable bit and appropriate global interrupt enable bit are set, the interrupt will vector immediately to address 000008h or 000018h, depending on the priority level. Individual interrupts can be disabled through their corresponding enable bits.

When the IPEN bit is cleared (default state), the interrupt priority feature is disabled and interrupts are compatible with PICmicro® mid-range devices. In Compatibility mode, the interrupt priority bits for each source have no effect. INTCON<6> is the PEIE bit, which enables/disables all peripheral interrupt sources. INTCON<7> is the GIE bit, which enables/disables all interrupt sources. All interrupts branch to address 000008h in Compatibility mode.

When an interrupt is responded to, the Global Interrupt Enable bit is cleared to disable further interrupts. If the IPEN bit is cleared, this is the GIE bit. If interrupt priority levels are used, this will be either the GIEH or GIEL bit. High priority interrupt sources can interrupt a low priority interrupt.

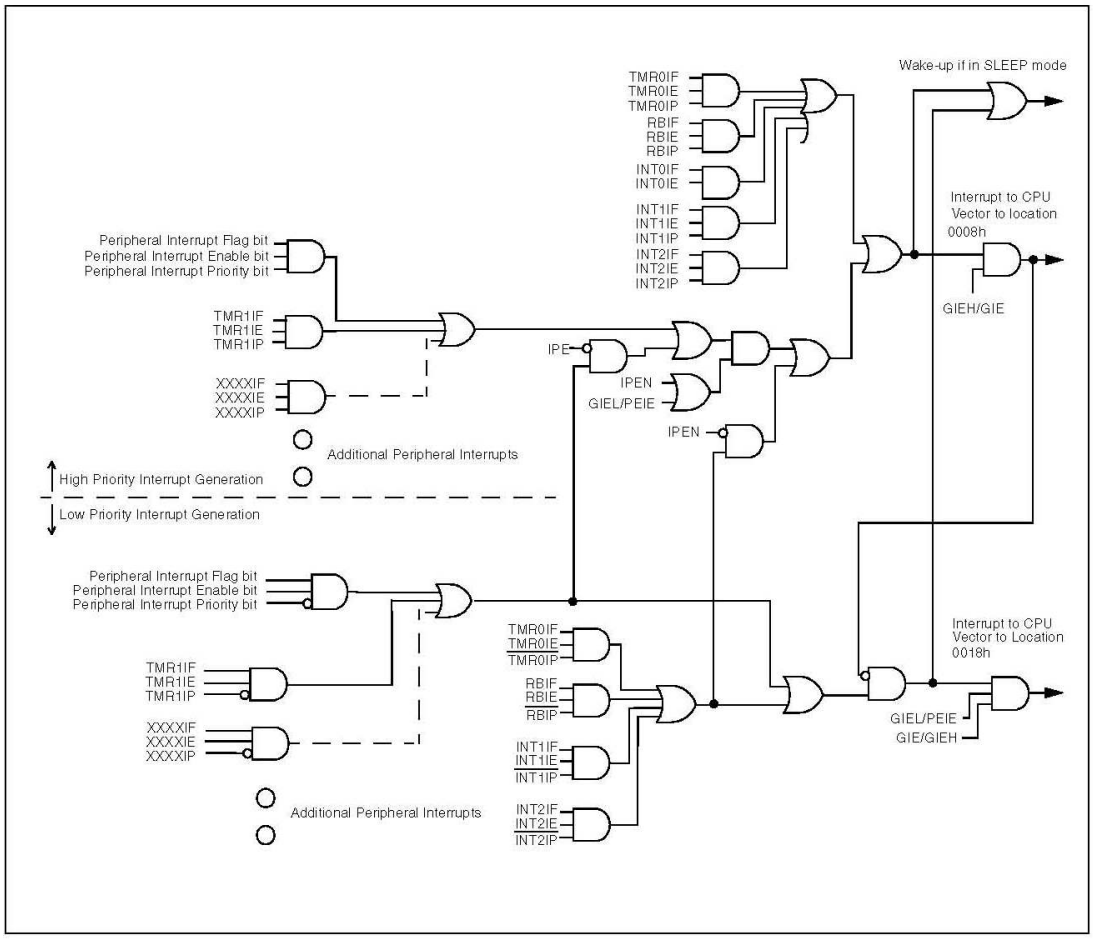
The return address is pushed onto the stack and the PC is loaded with the interrupt vector address (000008h or 000018h). Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bits must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

The "return from interrupt" instruction, `RETFIE`, exits the interrupt routine and sets the GIE bit (GIEH or GIEL if priority levels are used), which re-enables interrupts.

For external interrupt events, such as the INT pins or the PORTB input change interrupt, the interrupt latency will be three to four instruction cycles. The exact latency is the same for one or two-cycle instructions. Individual interrupt flag bits are set, regardless of the status of their corresponding enable bit or the GIE bit.

Note:	Do not use the `MOVFF` instruction to modify any of the Interrupt control registers while **any** interrupt is enabled. Doing so may cause erratic microcontroller behavior.

FIGURE 8-1: INTERRUPT LOGIC



© 2006 Microchip Technology Inc.

8.1 INTCON Registers

The INTCON Registers are readable and writable registers, which contain various enable, priority and flag bits.

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

REGISTER 8-1: INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 8-2: INTCON2 REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = All PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG0**:External Interrupt0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5 **INTEDG1**: External Interrupt1 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 4 **INTEDG2**: External Interrupt2 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 3 **Unimplemented:** Read as '0'

bit 2 **TMR0IP**: TMR0 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **Unimplemented:** Read as '0'

bit 0 **RBIP**: RB Port Change Interrupt Priority bit
1 = High priority
0 = Low priority

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

© 2006 Microchip Technology Inc.

REGISTER 8-3: INTCON3 REGISTER

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

bit 7 **INT2IP:** INT2 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **INT1IP:** INT1 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **Unimplemented:** Read as '0'

bit 4 **INT2IE:** INT2 External Interrupt Enable bit
1 = Enables the INT2 external interrupt
0 = Disables the INT2 external interrupt

bit 3 **INT1IE:** INT1 External Interrupt Enable bit
1 = Enables the INT1 external interrupt
0 = Disables the INT1 external interrupt

bit 2 **Unimplemented:** Read as '0'

bit 1 **INT2IF:** INT2 External Interrupt Flag bit
1 = The INT2 external interrupt occurred (must be cleared in software)
0 = The INT2 external interrupt did not occur

bit 0 **INT1IF:** INT1 External Interrupt Flag bit
1 = The INT1 external interrupt occurred (must be cleared in software)
0 = The INT1 external interrupt did not occur

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

8.3 PIE Registers

The PIE registers contain the individual enable bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Enable Registers (PIE1, PIE2). When IPEN = 0, the PEIE bit must be set to enable any of these peripheral interrupts.

REGISTER 8-6: PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **PSPIE(1):** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Master Synchronous Serial Port Interrupt Enable bit
1 = Enables the MSSP interrupt
0 = Disables the MSSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: This bit is reserved on PIC18F2X2 devices; always maintain this bit clear.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

© 2006 Microchip Technology Inc.

REGISTER 8-7: PIE2: PERIPHERAL INTERRUPT ENABLE REGISTER 2

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	EEIE	BCLIE	LVDIE	TMR3IE	CCP2IE
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **EEIE**: Data EEPROM/FLASH Write Operation Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 3 **BCLIE**: Bus Collision Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 2 **LVDIE**: Low Voltage Detect Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 1 **TMR3IE**: TMR3 Overflow Interrupt Enable bit
1 = Enables the TMR3 overflow interrupt
0 = Disables the TMR3 overflow interrupt

bit 0 **CCP2IE**: CCP2 Interrupt Enable bit
1 = Enables the CCP2 interrupt
0 = Disables the CCP2 interrupt

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

8.4 IPR Registers

The IPR registers contain the individual priority bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Priority Registers (IPR1, IPR2). The operation of the priority bits requires that the Interrupt Priority Enable (IPEN) bit be set.

REGISTER 8-8: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
bit 7							bit 0

bit 7 **PSPIP(1):** Parallel Slave Port Read/Write Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **ADIP**: A/D Converter Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **RCIP**: USART Receive Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TXIP**: USART Transmit Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **SSPIP**: Master Synchronous Serial Port Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **CCP1IP**: CCP1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR2IP**: TMR2 to PR2 Match Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **TMR1IP**: TMR1 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

Note 1: This bit is reserved on PIC18F2X2 devices; always maintain this bit set.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

© 2006 Microchip Technology Inc.

REGISTER 8-9: IPR2: PERIPHERAL INTERRUPT PRIORITY REGISTER 2

U-0	U-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	—	—	EEIP	BCLIP	LVDIP	TMR3IP	CCP2IP
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **EEIP**: Data EEPROM/FLASH Write Operation Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **BCLIP**: Bus Collision Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **LVDIP**: Low Voltage Detect Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR3IP**: TMR3 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **CCP2IP**: CCP2 Interrupt Priority bit
1 = High priority
0 = Low priority

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

8.6 INT0 Interrupt

External interrupts on the RB0/INT0, RB1/INT1 and RB2/INT2 pins are edge triggered: either rising, if the corresponding INTEDGx bit is set in the INTCON2 register, or falling, if the INTEDGx bit is clear. When a valid edge appears on the RBx/INTx pin, the corresponding flag bit INTxF is set. This interrupt can be disabled by clearing the corresponding enable bit INTxE. Flag bit INTxF must be cleared in software in the Interrupt Service Routine before re-enabling the interrupt. All external interrupts (INT0, INT1 and INT2) can wake-up the processor from SLEEP, if bit INTxE was set prior to going into SLEEP. If the global interrupt enable bit GIE is set, the processor will branch to the interrupt vector following wake-up.

Interrupt priority for INT1 and INT2 is determined by the value contained in the interrupt priority bits, INT1IP (INTCON3<6>) and INT2IP (INTCON3<7>). There is no priority bit associated with INT0. It is always a high priority interrupt source.

8.7 TMR0 Interrupt

In 8-bit mode (which is the default), an overflow (FFh → 00h) in the TMR0 register will set flag bit TMR0IF. In 16-bit mode, an overflow (FFFFh → 0000h) in the TMR0H:TMR0L registers will set flag bit TMR0IF. The interrupt can be enabled/disabled by setting/clearing enable bit T0IE (INTCON<5>). Interrupt priority for Timer0 is determined by the value contained in the interrupt priority bit TMR0IP (INTCON2<2>). See Section 10.0 for further details on the Timer0 module.

8.8 PORTB Interrupt-on-Change

An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit, RBIE (INTCON<3>). Interrupt priority for PORTB interrupt-on-change is determined by the value contained in the interrupt priority bit, RBIP (INTCON2<0>).

8.9 Context Saving During Interrupts

During an interrupt, the return PC value is saved on the stack. Additionally, the WREG, STATUS and BSR registers are saved on the fast return stack. If a fast return from interrupt is not used (See Section 4.3), the user may need to save the WREG, STATUS and BSR registers in software. Depending on the user's application, other registers may also need to be saved. Equation 8-1 saves and restores the WREG, STATUS and BSR registers during an Interrupt Service Routine.

EXAMPLE 8-1: SAVING STATUS, WREG AND BSR REGISTERS IN RAM

```
MOVWF   W_TEMP                          ; W_TEMP is in virtual bank
MOVFF   STATUS, STATUS_TEMP             ; STATUS_TEMP located anywhere
MOVFF   BSR,    BSR_TEMP                ; BSR located anywhere
;
; USER ISR CODE
;
MOVFF   BSR_TEMP,   BSR                 ; Restore BSR
MOVF    W_TEMP,     W                   ; Restore WREG
MOVFF   STATUS_TEMP,STATUS              ; Restore STATUS
```

10.0 TIMER0 MODULE

The Timer0 module has the following features:

- Software selectable as an 8-bit or 16-bit timer/counter
- Readable and writable
- Dedicated 8-bit software programmable prescaler
- Clock source selectable to be external or internal
- Interrupt-on-overflow from FFh to 00h in 8-bit mode and FFFFh to 0000h in 16-bit mode
- Edge select for external clock

Figure 10-1 shows a simplified block diagram of the Timer0 module in 8-bit mode and Figure 10-2 shows a simplified block diagram of the Timer0 module in 16-bit mode.

The T0CON register (Register 10-1) is a readable and writable register that controls all the aspects of Timer0, including the prescale selection.

REGISTER 10-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

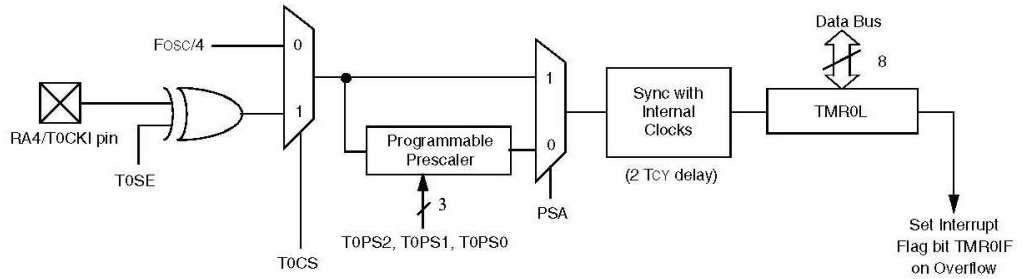
bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 prescale value
110 = 1:128 prescale value
101 = 1:64 prescale value
100 = 1:32 prescale value
011 = 1:16 prescale value
010 = 1:8 prescale value
001 = 1:4 prescale value
000 = 1:2 prescale value

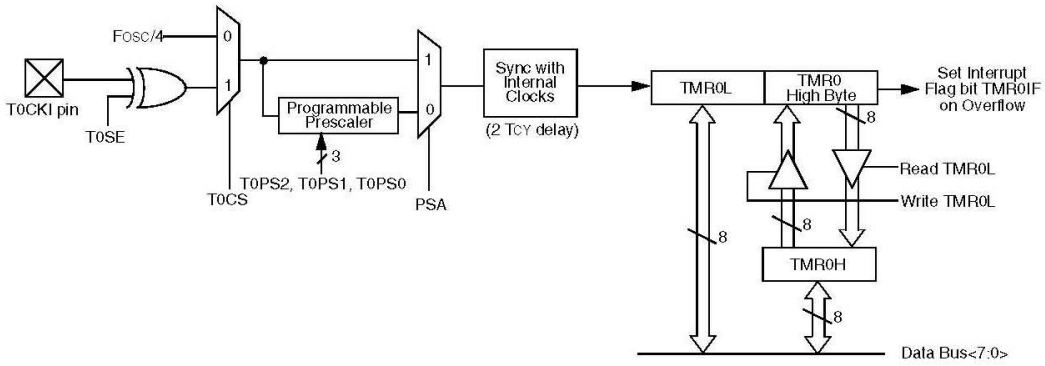
Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

FIGURE 10-1: TIMER0 BLOCK DIAGRAM IN 8-BIT MODE



Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

FIGURE 10-2: TIMER0 BLOCK DIAGRAM IN 16-BIT MODE



Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

© 2006 Microchip Technology Inc.

10.1 Timer0 Operation

Timer0 can operate as a timer or as a counter.

Timer mode is selected by clearing the T0CS bit. In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0L register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0L register.

Counter mode is selected by setting the T0CS bit. In Counter mode, Timer0 will increment, either on every rising or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit (T0SE). Clearing the T0SE bit selects the rising edge. Restrictions on the external clock input are discussed below.

When an external clock input is used for Timer0, it must meet certain requirements. The requirements ensure the external clock can be synchronized with the internal phase clock (Tosc). Also, there is a delay in the actual incrementing of Timer0 after synchronization.

10.2 Prescaler

An 8-bit counter is available as a prescaler for the Timer0 module. The prescaler is not readable or writable.

The PSA and T0PS2:T0PS0 bits determine the prescaler assignment and prescale ratio.

Clearing bit PSA will assign the prescaler to the Timer0 module. When the prescaler is assigned to the Timer0 module, prescale values of 1:2, 1:4,..., 1:256 are selectable.

When assigned to the Timer0 module, all instructions writing to the TMR0L register (e.g., `CLRF TMR0`, `MOVWF TMR0`, `BSF TMR0, x`....etc.) will clear the prescaler count.

Note:	Writing to TMR0L when the prescaler is assigned to Timer0 will clear the prescaler count, but will not change the prescaler assignment.

10.2.1 SWITCHING PRESCALER ASSIGNMENT

The prescaler assignment is fully under software control, (i.e., it can be changed "on-the-fly" during program execution).

10.3 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h in 8-bit mode, or FFFFh to 0000h in 16-bit mode. This overflow sets the TMR0IF bit. The interrupt can be masked by clearing the TMR0IE bit. The TMR0IE bit must be cleared in software by the Timer0 module Interrupt Service Routine before re-enabling this interrupt. The TMR0 interrupt cannot awaken the processor from SLEEP, since the timer is shut-off during SLEEP.

10.4 16-Bit Mode Timer Reads and Writes

TMR0H is not the high byte of the timer/counter in 16-bit mode, but is actually a buffered version of the high byte of Timer0 (refer to Figure 10-2). The high byte of the Timer0 counter/timer is not directly readable nor writable. TMR0H is updated with the contents of the high byte of Timer0 during a read of TMR0L. This provides the ability to read all 16-bits of Timer0 without having to verify that the read of the high and low byte were valid due to a rollover between successive reads of the high and low byte.

A write to the high byte of Timer0 must also take place through the TMR0H buffer register. Timer0 high byte is updated with the contents of TMR0H when a write occurs to TMR0L. This allows all 16-bits of Timer0 to be updated at once.

TABLE 10-1: REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
TMR0L	Timer0 Module Low Byte Register								xxxx xxxx	uuuu uuuu
TMR0H	Timer0 Module High Byte Register								0000 0000	0000 0000
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	1111 1111	1111 1111
TRISA	—	PORTA Data Direction Register							-111 1111	-111 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

11.0 TIMER1 MODULE

The Timer1 module timer/counter has the following features:

- 16-bit timer/counter
  (two 8-bit registers; TMR1H and TMR1L)
- Readable and writable (both registers)
- Internal or external clock select
- Interrupt-on-overflow from FFFFh to 0000h
- RESET from CCP module special event trigger

Figure 11-1 is a simplified block diagram of the Timer1 module.

Register 11-1 details the Timer1 control register. This register controls the Operating mode of the Timer1 module, and contains the Timer1 oscillator enable bit (T1OSCEN). Timer1 can be enabled or disabled by setting or clearing control bit TMR1ON (T1CON<0>).

REGISTER 11-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable bit
1 = Enables register Read/Write of Timer1 in one 16-bit operation
0 = Enables register Read/Write of Timer1 in two 8-bit operations

bit 6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit
1 = Timer1 Oscillator is enabled
0 = Timer1 Oscillator is shut-off
The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Select bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T13CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

11.1 Timer1 Operation

Timer1 can operate in one of these modes:

- As a timer
- As a synchronous counter
- As an asynchronous counter

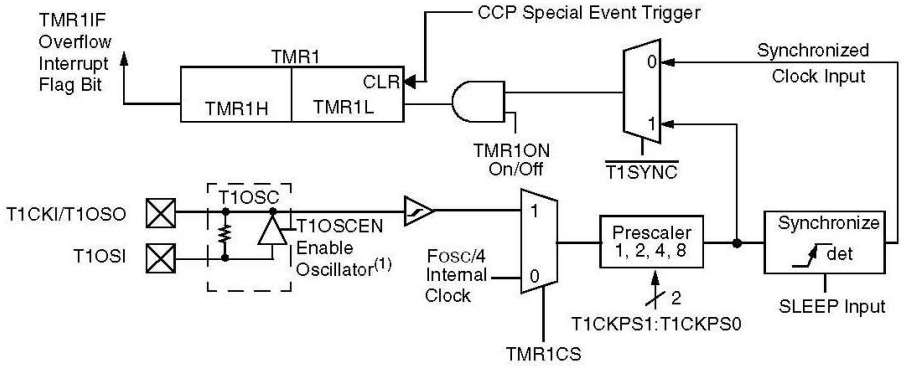
The Operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

When TMR1CS = 0, Timer1 increments every instruction cycle. When TMR1CS = 1, Timer1 increments on every rising edge of the external clock input or the Timer1 oscillator, if enabled.

When the Timer1 oscillator is enabled (T1OSCEN is set), the RC1/T1OSI and RC0/T1OSO/T1CKI pins become inputs. That is, the TRISC<1:0> value is ignored, and the pins are read as '0'.

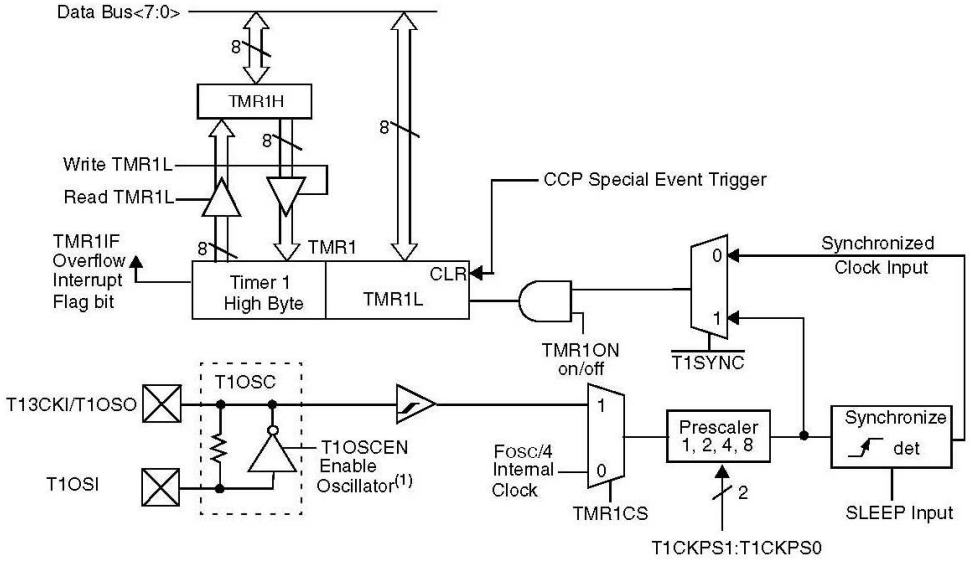
Timer1 also has an internal "RESET input". This RESET can be generated by the CCP module (Section 14.0).

FIGURE 11-1: TIMER1 BLOCK DIAGRAM



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

FIGURE 11-2: TIMER1 BLOCK DIAGRAM: 16-BIT READ/WRITE MODE



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

© 2006 Microchip Technology Inc.

11.2 Timer1 Oscillator

A crystal oscillator circuit is built-in between pins T1OSI (input) and T1OSO (amplifier output). It is enabled by setting control bit T1OSCEN (T1CON<3>). The oscillator is a low power oscillator rated up to 200 kHz. It will continue to run during SLEEP. It is primarily intended for a 32 kHz crystal. Table 11-1 shows the capacitor selection for the Timer1 oscillator.

The user must provide a software time delay to ensure proper start-up of the Timer1 oscillator.

TABLE 11-1: CAPACITOR SELECTION FOR THE ALTERNATE OSCILLATOR

Osc Type	Freq	C1	C2
LP	32 kHz	TBD(1)	TBD(1)
Crystal to be Tested:			
32.768 kHz	Epson C-001R32.768K-A		± 20 PPM

Note 1: Microchip suggests 33 pF as a starting point in validating the oscillator circuit.

2: Higher capacitance increases the stability of the oscillator, but also increases the start-up time.

3: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.

4: Capacitor values are for design guidance only.

11.3 Timer1 Interrupt

The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 Interrupt, if enabled, is generated on overflow, which is latched in interrupt flag bit TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit, TMR1IE (PIE1<0>).

11.4 Resetting Timer1 using a CCP Trigger Output

If the CCP module is configured in Compare mode to generate a "special event trigger" (CCP1M3:CCP1M0 = 1011), this signal will reset Timer1 and start an A/D conversion (if the A/D module is enabled).

Note: The special event triggers from the CCP1 module will not set interrupt flag bit TMR1IF (PIR1<0>).

Timer1 must be configured for either Timer or Synchronized Counter mode to take advantage of this feature. If Timer1 is running in Asynchronous Counter mode, this RESET operation may not work.

In the event that a write to Timer1 coincides with a special event trigger from CCP1, the write will take precedence.

In this mode of operation, the CCPR1H:CCPR1L registers pair effectively becomes the period register for Timer1.

11.5 Timer1 16-Bit Read/Write Mode

Timer1 can be configured for 16-bit reads and writes (see Figure 11-2). When the RD16 control bit (T1CON<7>) is set, the address for TMR1H is mapped to a buffer register for the high byte of Timer1. A read from TMR1L will load the contents of the high byte of Timer1 into the Timer1 high byte buffer. This provides the user with the ability to accurately read all 16-bits of Timer1 without having to determine whether a read of the high byte followed by a read of the low byte is valid, due to a rollover between reads.

A write to the high byte of Timer1 must also take place through the TMR1H buffer register. Timer1 high byte is updated with the contents of TMR1H when a write occurs to TMR1L. This allows a user to write all 16 bits to both the high and low bytes of Timer1 at once.

The high byte of Timer1 is not directly readable or writable in this mode. All reads and writes must take place through the Timer1 high byte buffer register. Writes to TMR1H do not clear the Timer1 prescaler. The prescaler is only cleared on writes to TMR1L.

TABLE 11-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	0000 0000	0000 0000
TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
T1CON	RD16	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	0-00 0000	u-uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X2 devices; always maintain these bits clear.

© 2006 Microchip Technology Inc.